Overview

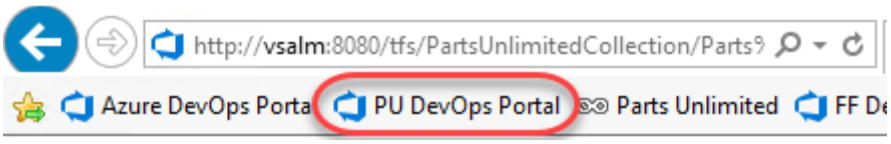
In this lab, you will learn how to use Azure Pipelines to setup a continuous integration (CI) pipeline to build and test your applications. This scriptable build system is both web-based and cross-platform, while also providing a modern interface for visualizing sophisticated workflows. Although we won't demonstrate all of the cross-platform possibilities in this lab, it is important to note that you can also build for iOS, Android, Java (using Ant, Maven, or Gradle), and Linux.

Exercise 1: Build Agent Pools and Queues

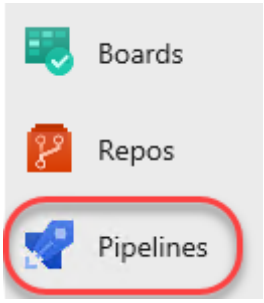
In this exercise, you will learn how to create and configure build agent pools and queues in order to support the new agents in Azure DevOps Server 2019. This new scriptable build system is web-based and cross-platform, and is recommended for all new and existing builds going forward.

Task 1: Getting to build pipelines in Azure DevOps

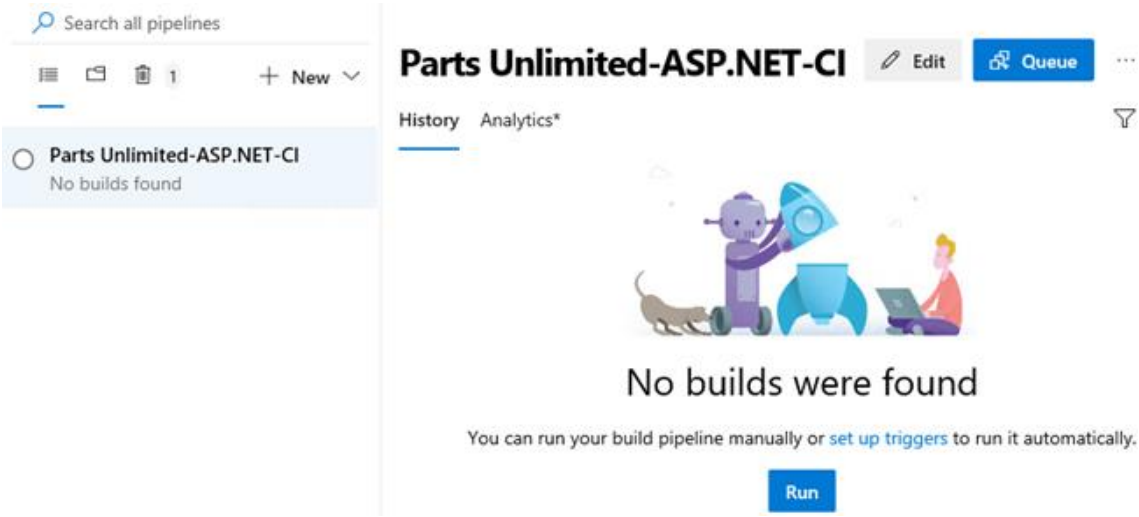
1. Log in
2. Let's get started by touring the **Build** hub in the web portal. Launch **Internet Explorer** from the taskbar and click **PU DevOps Portal** from the favorites bar at the top.



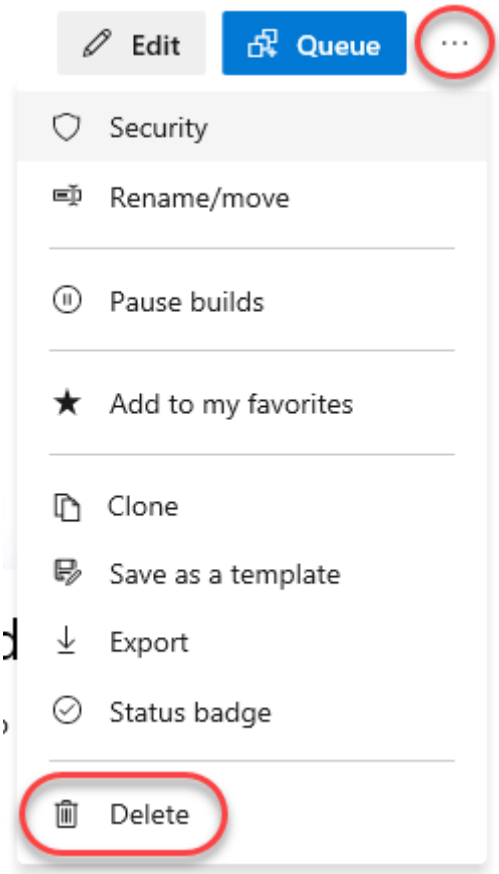
3. Navigate to the **Pipelines** hub.



4. The default view lists build pipelines with the first pipeline selected. The VM ships with a default pipeline, although there are no builds in the history.

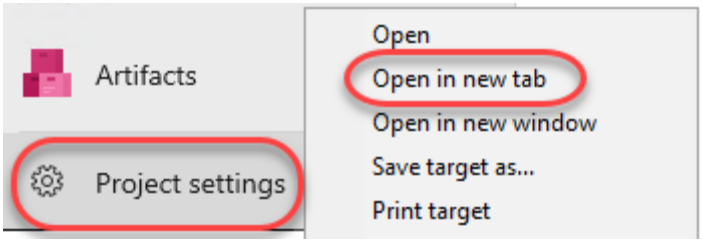


5. From the build pipeline dropdown, select **Delete** and confirm the delete. This lab will focusing on rebuilding this pipeline.

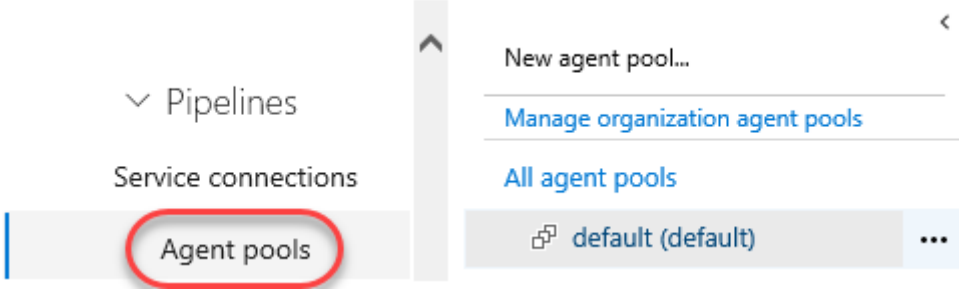


Task 2: Creating an agent pool

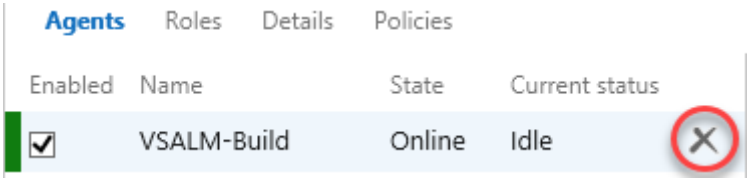
1. The first thing that we need to do is to set up an agent pool for the project. This pool can contain both Windows and cross-platform agents. Right-click **Project settings** and select **Open in new tab**.



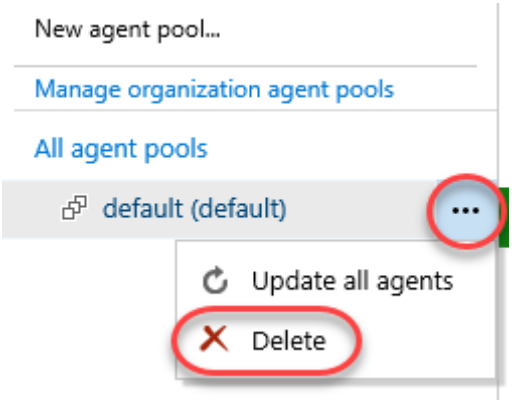
2. Navigate to **Agent pools** under **Pipelines**.



3. There is already an agent pool named **"default"** with a single agent as shown here. For the purposes of this lab, we will delete this pool and add it back in with a new agent. **Delete** the agent.



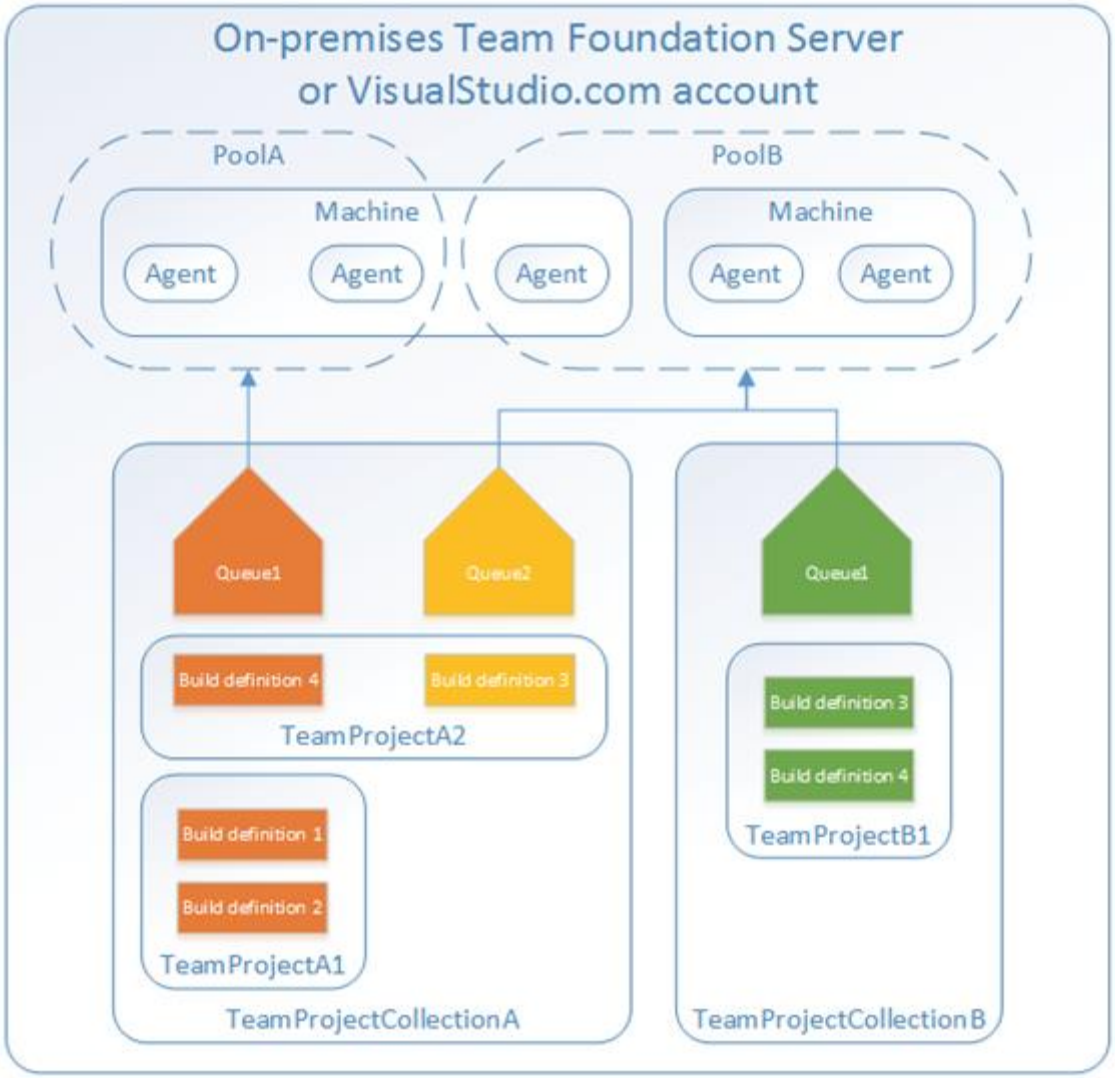
4. **Delete** the pool and confirm.



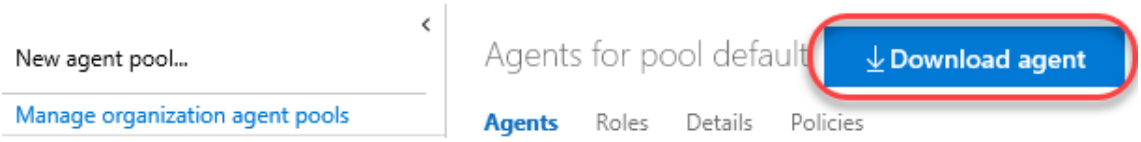
Task 3: Creating a build queue

1. Before we continue with the installation of an agent, let's also ensure that we set up our team project collection with a build **queue** that points to the default agent pool. Since queues are scoped to your team project collection, you can share them across build definitions and team projects.

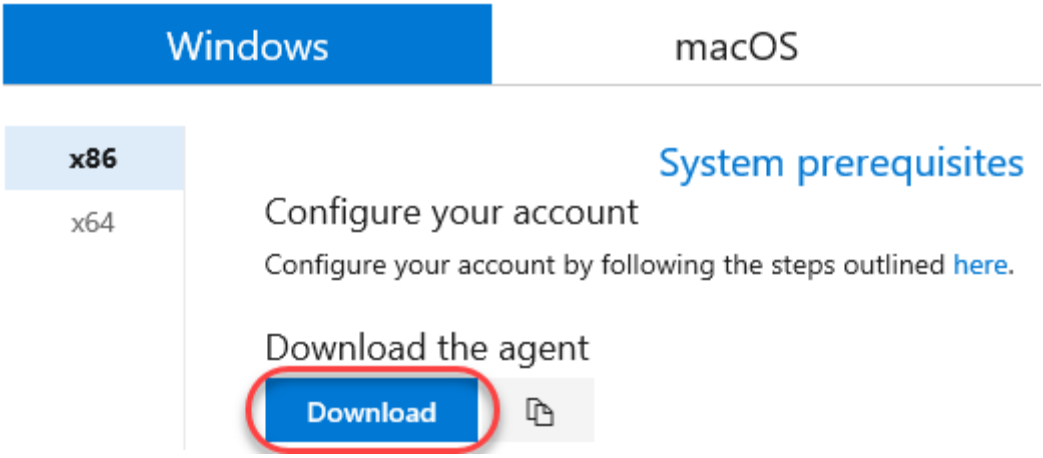
2. This diagram from the MSDN documentation helps to illustrate the relationship between pools, queues, team project collections, and build definitions. Note that you can also install multiple agents on a single machine.



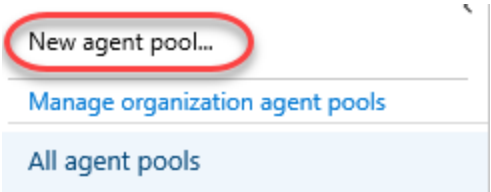
3. Click **Download agent**.



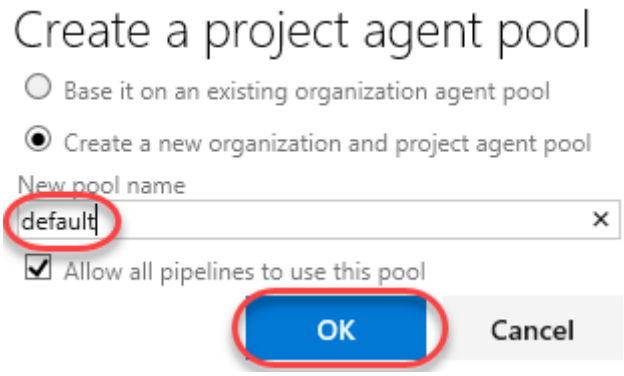
4. Click **Download** and save the target to disk in a convenient place. This download may take a few minutes, so you can close the **Download agent** dialog and continue to the next step.



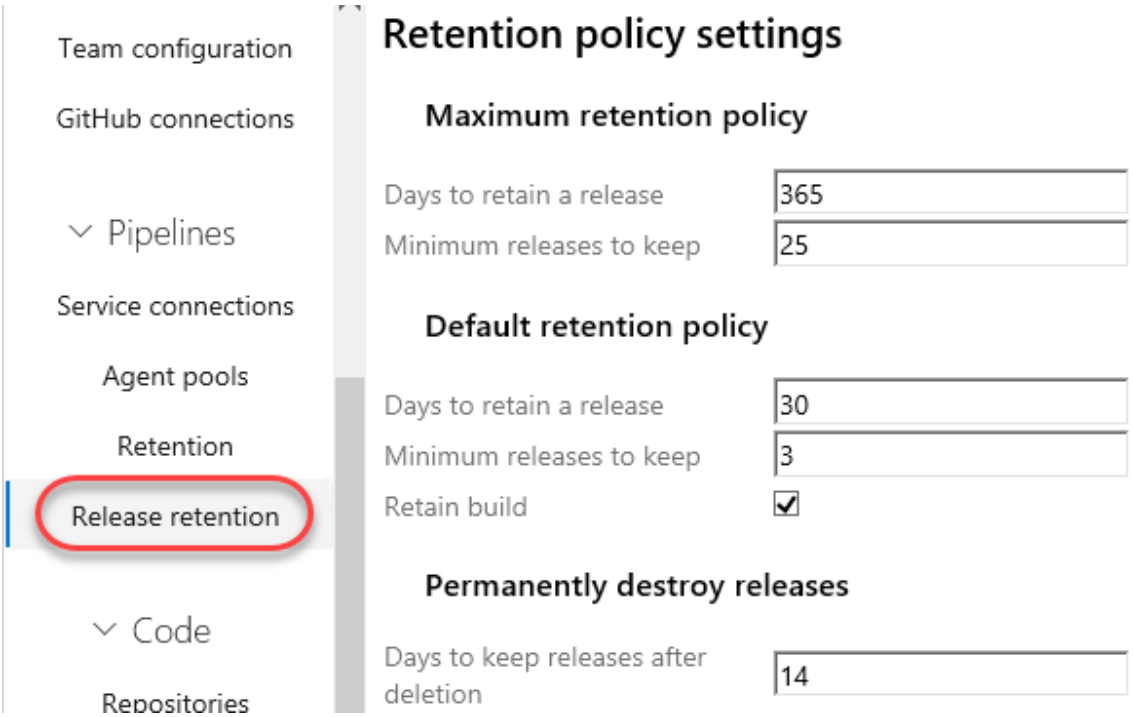
5. Click **New agent pool** to create a new pool.



6. Enter a **New pool name** of "**default**" and click **OK**.

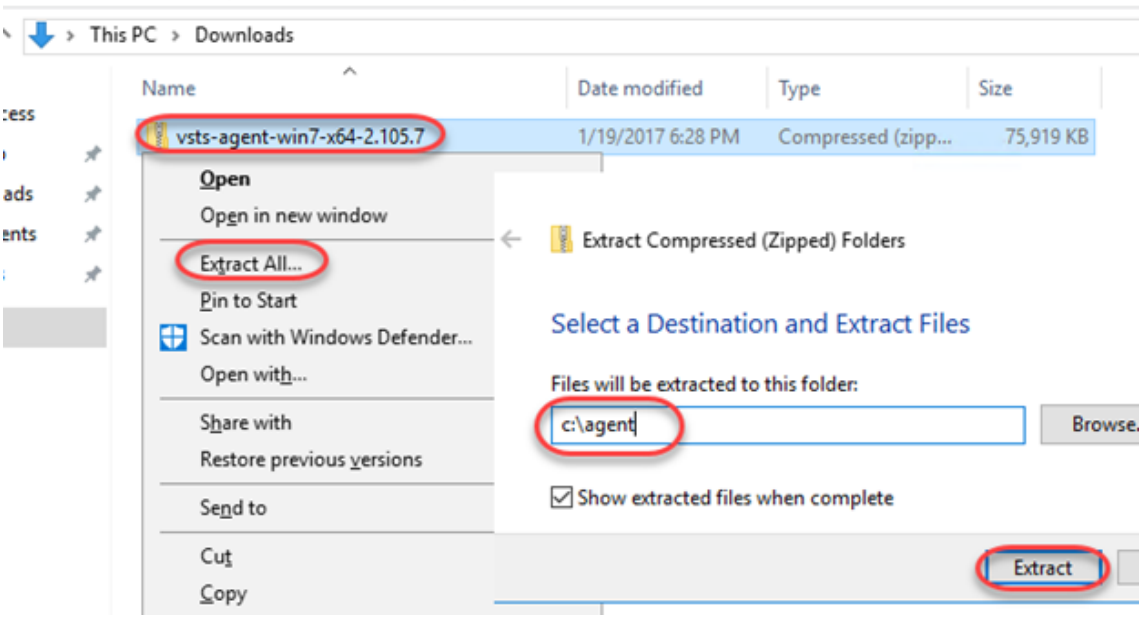


7. Select the **Release retention** tab. From here, you can specify the default and maximum settings for how long the system retains completed builds. The default retention policy is set at 30 days, with the maximum at 365 days. This means that regardless of what is set on the individual build definition all builds that have not been marked to "Retain indefinitely" will be deleted 30 days after they complete.



Task 4: Installing and configuring an agent

1. Wait for the agent download to finish if it has not already. Unzip it to **c:\agent** when complete.



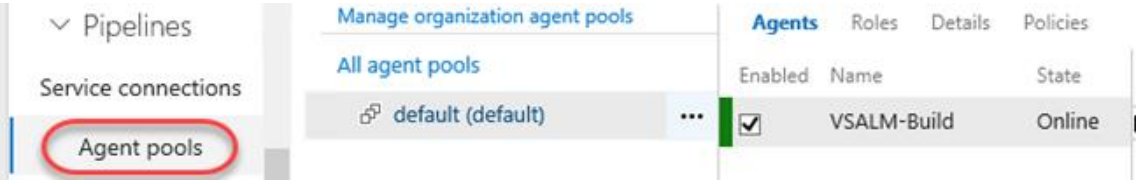
2. Launch an instance of **Command Prompt** as **Administrator** from the taskbar.
3. Change to the unzipped agent directory.
4. `cd c:\agent`
5. Execute the agent configuration script.
6. `config.cmd`
7. Enter the server URL "http://vsalm:8080/tfs".
8. Press **Enter** for **Integrated** authentication.

9. Press **Enter** to use the default agent pool of "default".
10. Set the agent name to **"VSALM-Build"** and press **Enter**.
11. Press **Enter** to use the default path proposed for the agent work folder "c:\agent_work".
12. When asked if you want to install as a Windows Service, type **"Y"** and then press **Enter**. Note that you could also configure the agent to run in interactive mode, which you may want to do if you were planning to execute coded UI tests.
13. Press **Enter** to run as network service, rather than providing a specific user account.
14. After a few moments, the script should complete with the successful installation and configuration of the new agent.

```
C:\agent>config

>> Connect:

Enter server URL > http://vsalm:8080/tfs
Enter authentication type (press enter for Integrated) >
Connecting to server ...

>> Register Agent:

Enter agent pool (press enter for default) >
Enter agent name (press enter for VSALM) > VSALM-Build
Scanning for tool capabilities.
Connecting to the server.
Successfully added the agent
Testing agent connection.
Enter work folder (press enter for _work) >
2019-03-20 19:31:41Z: Settings Saved.
Enter run agent as service? (Y/N) (press enter for N) > y
Enter User account to use for the service (press enter for NT AUTHORITY\NETWORK SERVICE) >
Granting file permissions to 'NT AUTHORITY\NETWORK SERVICE'.
Service vstsagent.vsalm.VSALM-Build successfully installed
Service vstsagent.vsalm.VSALM-Build successfully set recovery option
Service vstsagent.vsalm.VSALM-Build successfully set to delayed auto start
Service vstsagent.vsalm.VSALM-Build successfully configured
Service vstsagent.vsalm.VSALM-Build started successfully

C:\agent>_
```

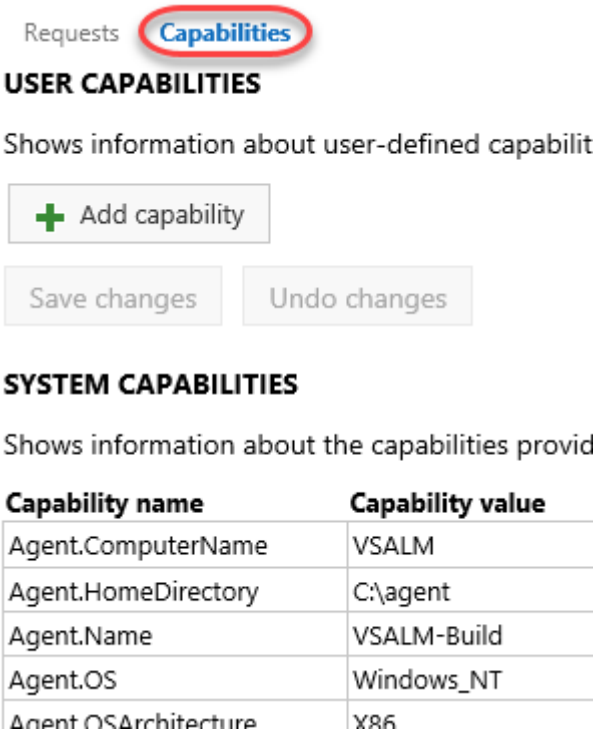
> **Note:** You weren't prompted for credentials in this case, but under normal circumstances when installing on a remote machine you would be asked to sign in as an agent pool administrator. These credentials are only used once during the configuration process.

15. Return to **Internet Explorer** and navigate to the **Agent pools** tab. The newly created agent will now be in the pool.



16. Select the **Capabilities** tab to take note of the System Capabilities list shown for the agent. System capabilities are name/value pairs that you can use to ensure that your build definition is only run by build agents that meet specified criteria. Environment variables automatically appear in the list. Some additional capabilities (such as .NET Frameworks) are also added automatically. You can also add your own

capabilities to the list based on additional requirements for your builds. Later, when a build is queued, the system sends the job only to agents that have the capabilities demanded by the build definition.



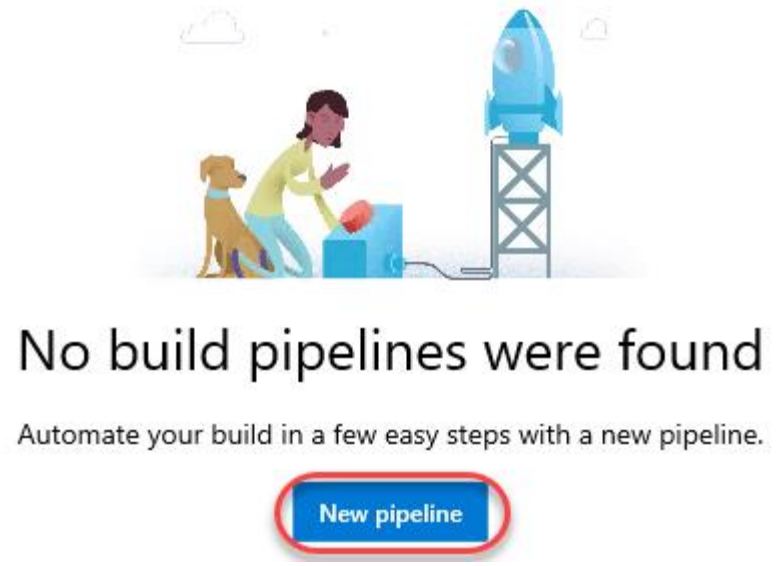
17. Close the browser tab to return to the build pipeline page.

Exercise 2: Working with build pipelines

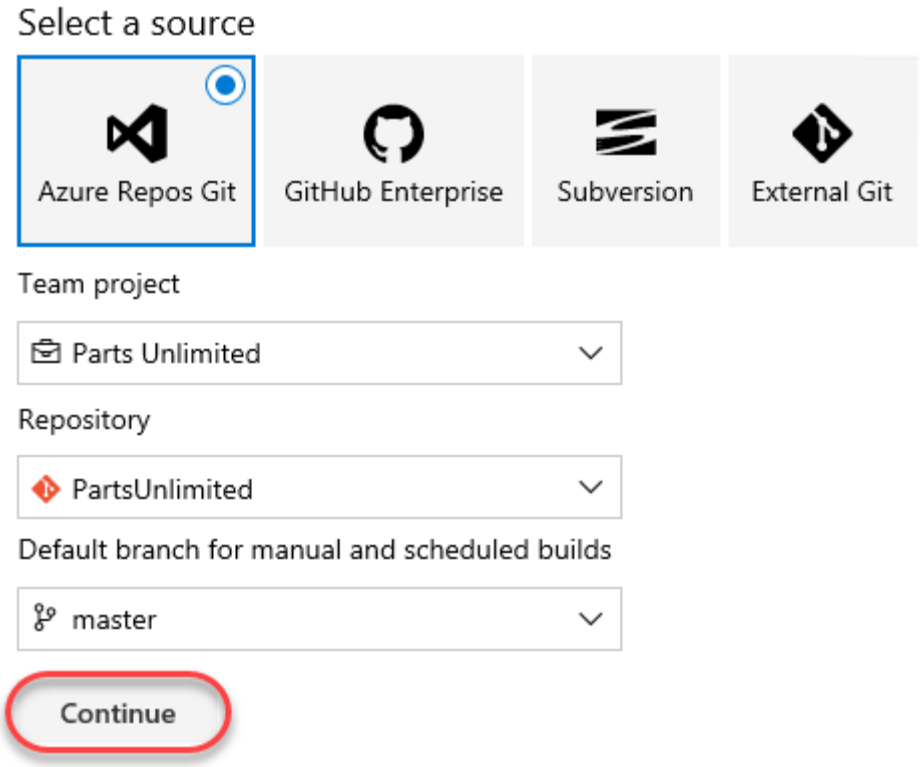
In this exercise, you will learn how to create a basic build definition from one of the provided templates and then queue the build for execution.

Task 1: Creating a basic build definition from a template

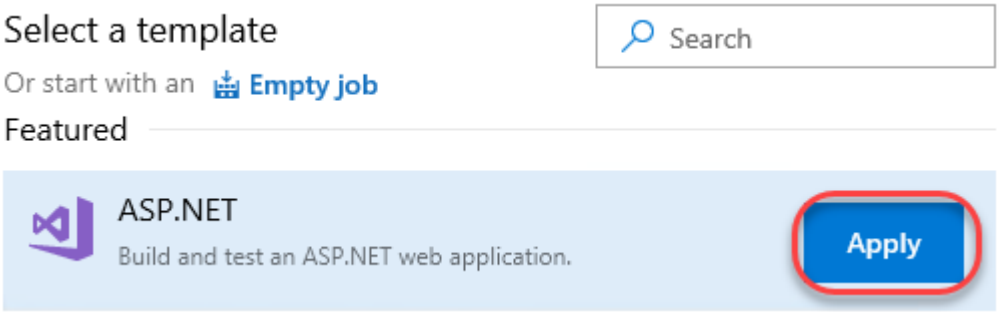
1. Click **New pipeline** to create a new build pipeline.



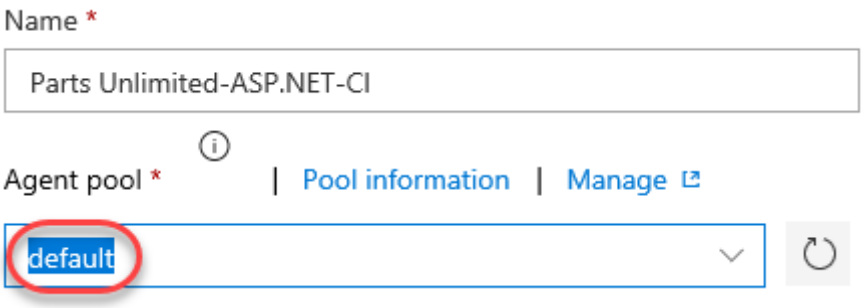
2. There are a lot of options for selecting the build source, team project, and repo. Accept the defaults and click **Continue**. This will build the master branch of the PartsUnlimited project.



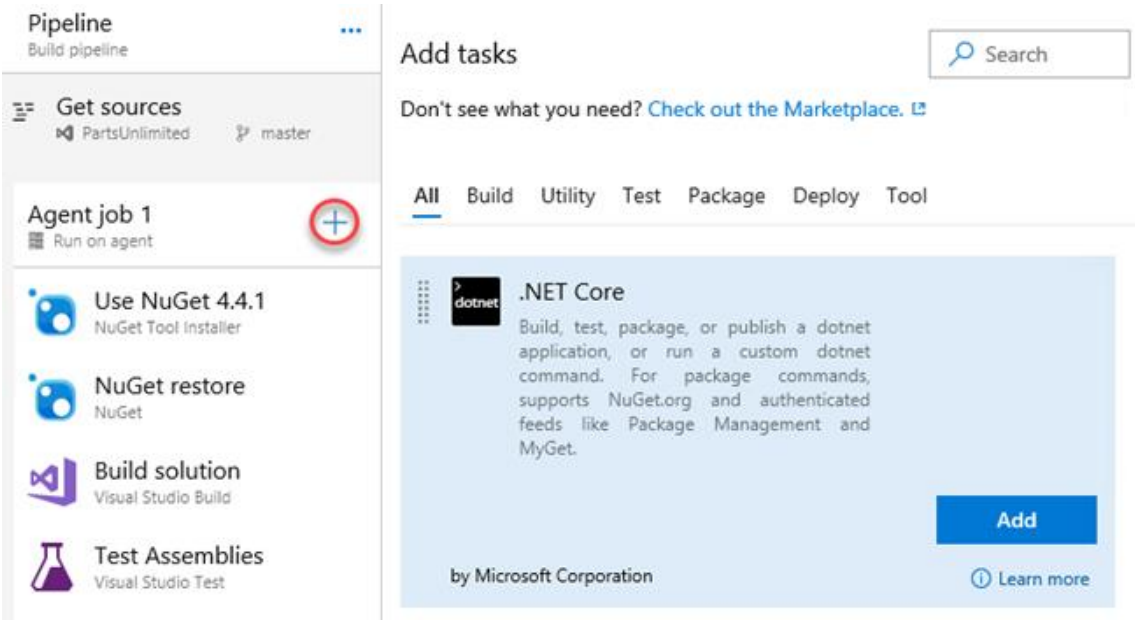
3. There are many templates available to build common project types. Everything is customizable, and you can even start with an empty pipeline. Locate the **ASP.NET** template and click **Apply**.



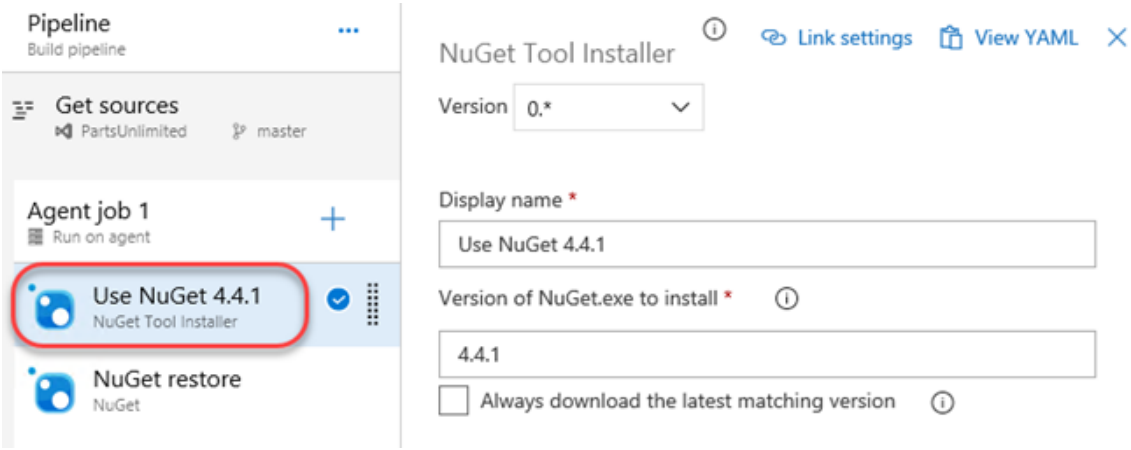
4. For **Agent pool**, select **default** to use the pool created earlier.



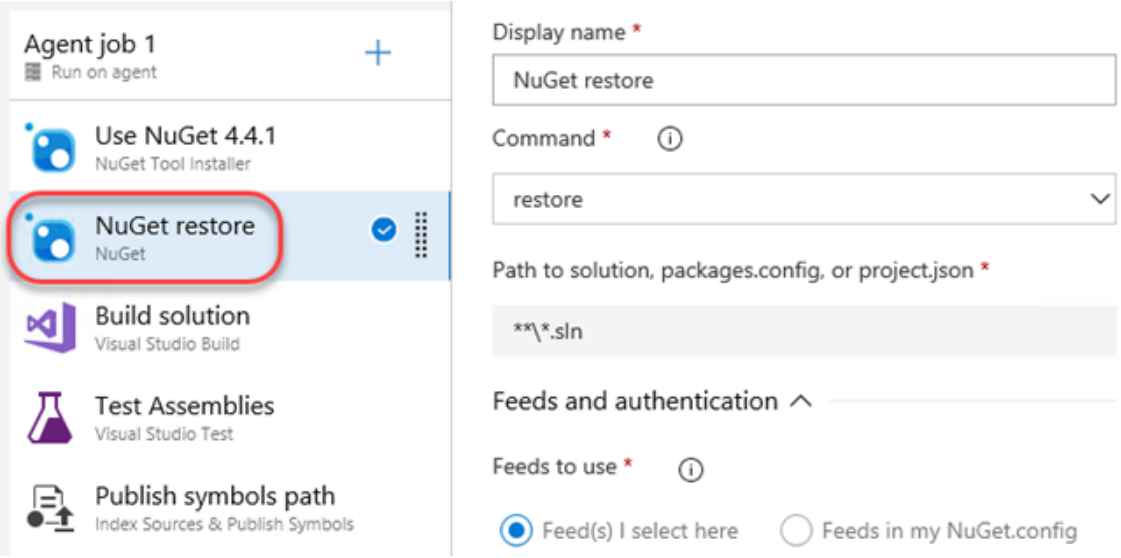
5. Click the **Add task** button. You can find a wide variety of tasks to cover common scenarios from build through deploy and everything in between. Do not add another task at this time.



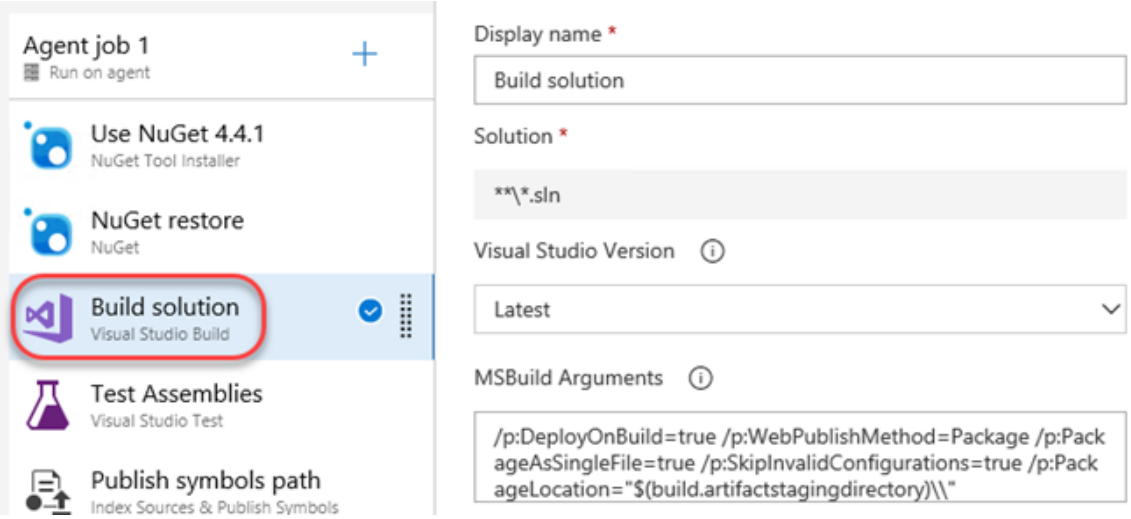
6. Select the **Use NuGet** task. This will ensure the specified version of NuGet is installed.



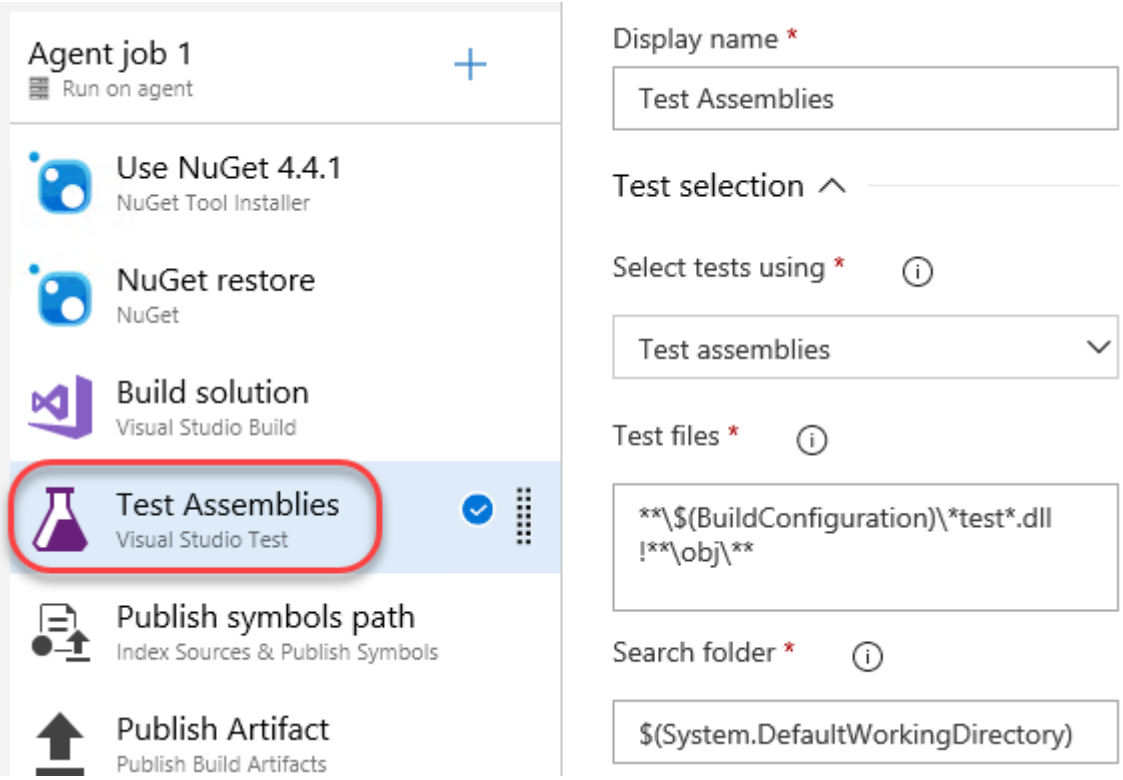
7. Select **NuGet restore**. This task restores all NuGet packages required by the solution.



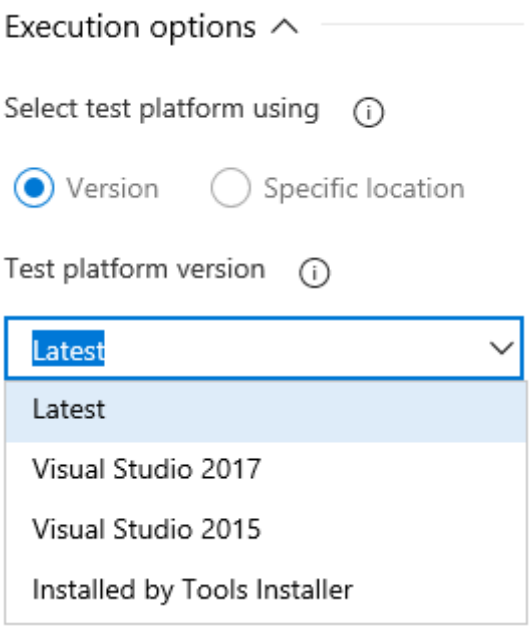
8. Select **Build solution**. This will build the solution using the default parameters specified by the ASP.NET template selected earlier.



9. Select **Test Assemblies**. This task runs project tests based on the configuration. By default, they are detected in the assemblies based on the pattern shown.

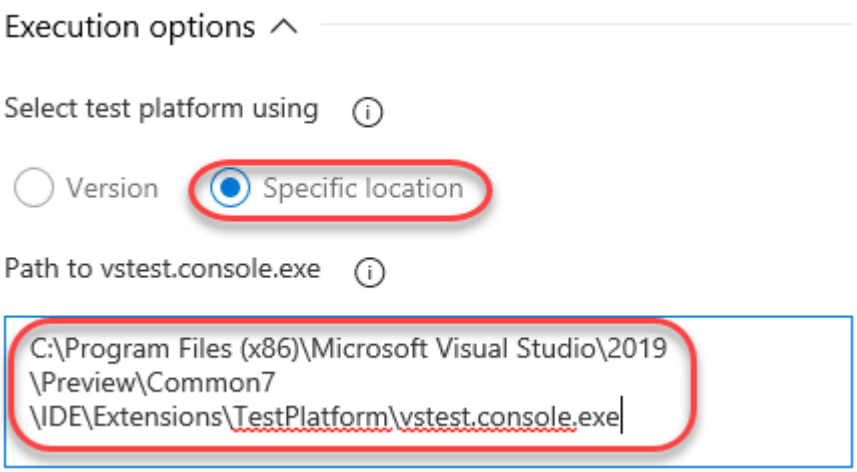


10. From the **Test platform version**, check for the version of Visual Studio installed. If it is available, select it.

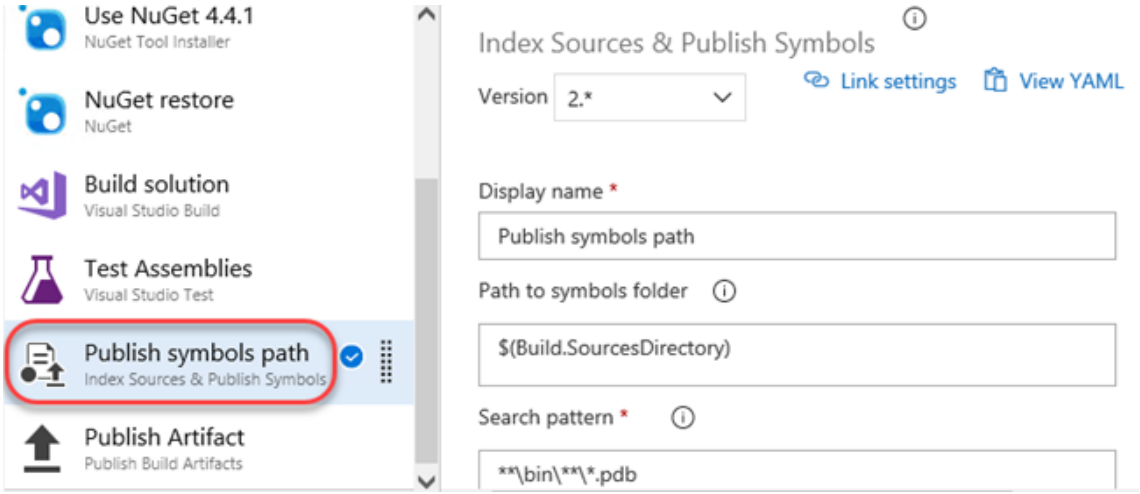


11. If the installed version of Visual Studio is not available, select **Specific location** and enter the path to **vstest.console.exe**. It should be something like **"C:\Program Files (x86)\Microsoft Visual**

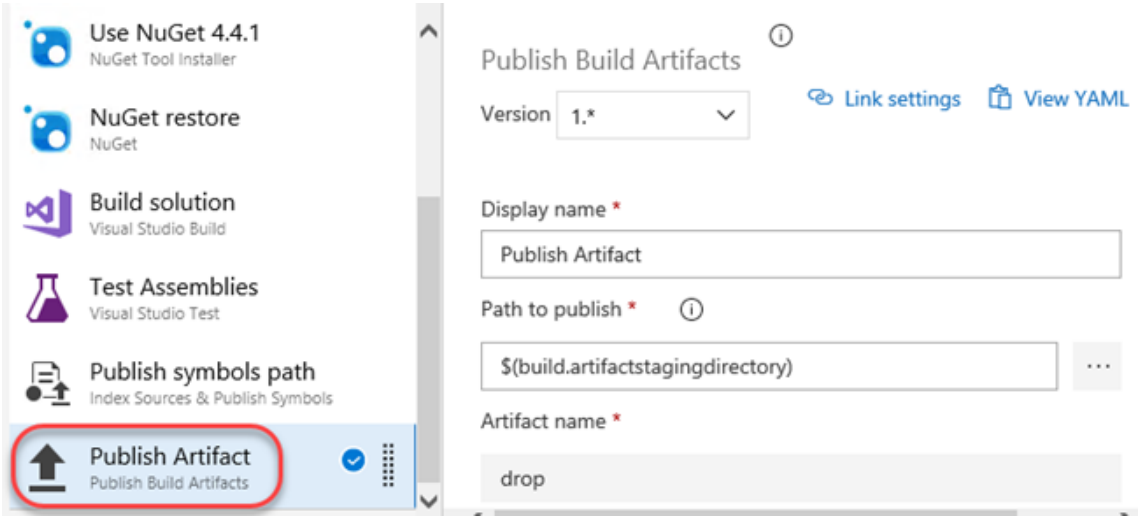
Studio\2019\Preview\Common7\IDE\Extensions\TestPlatform\vstest.console.exe".



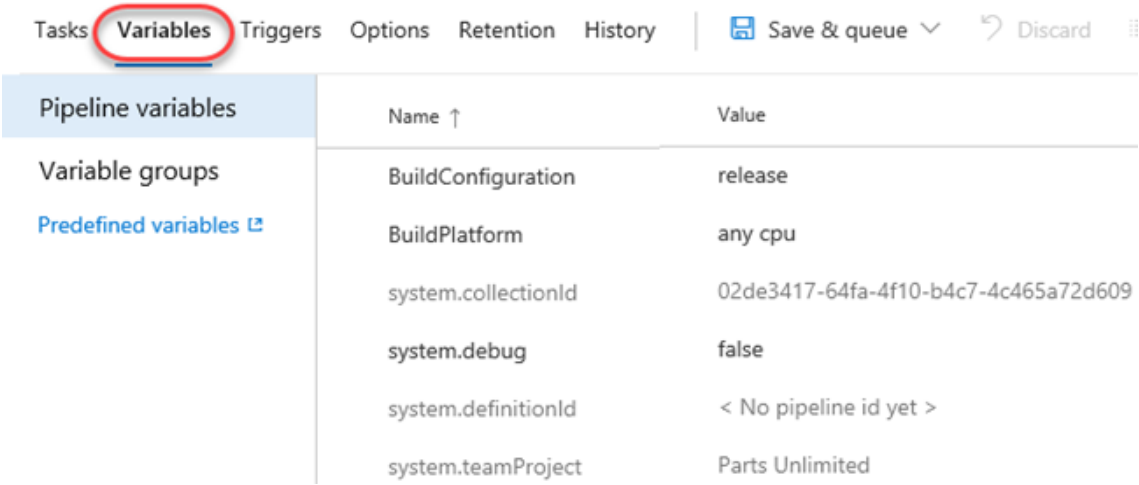
12. Select **Publish symbols path**. This task specifies where and how symbols are pushed.



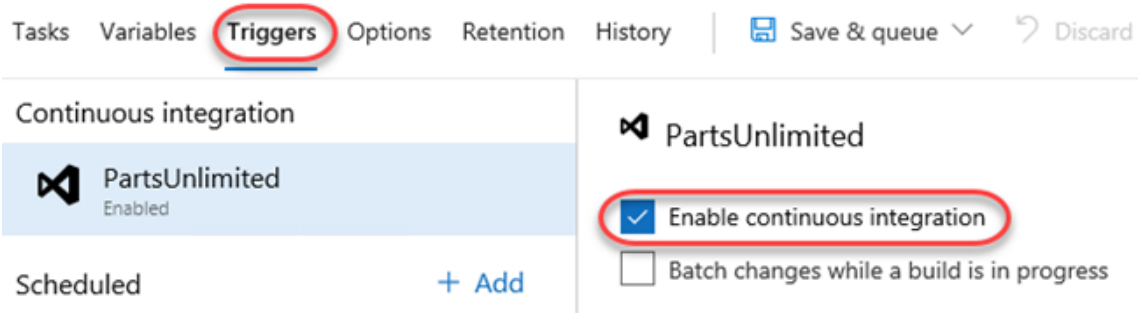
13. Select **Publish Artifact**. This task specifies where and how the project artifact is published.



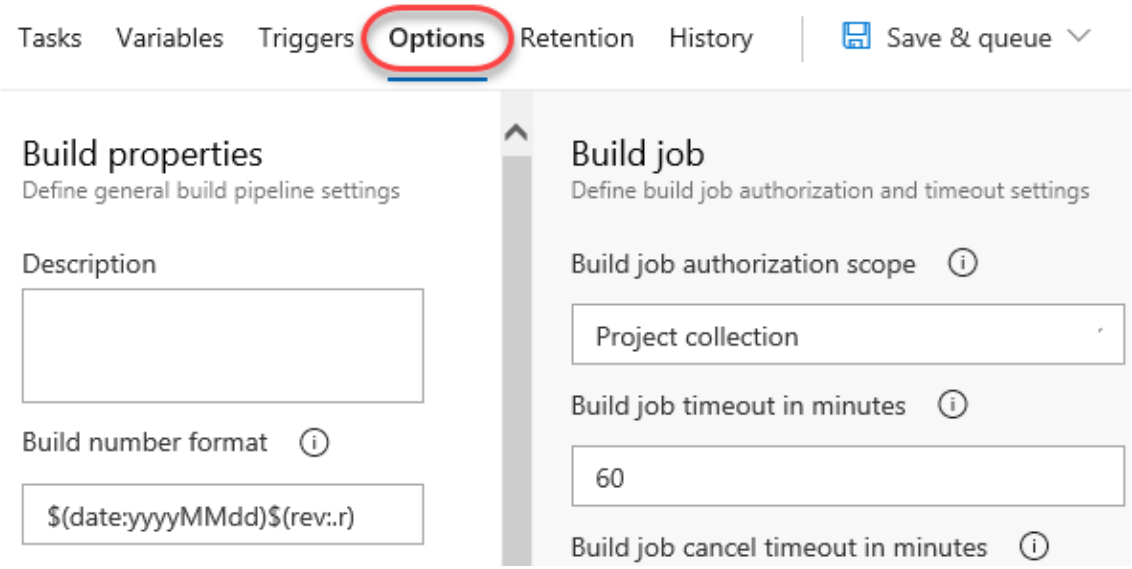
14. Select the **Variables** tab. This enables you to specify centralized variables to share with the tasks of the pipeline.



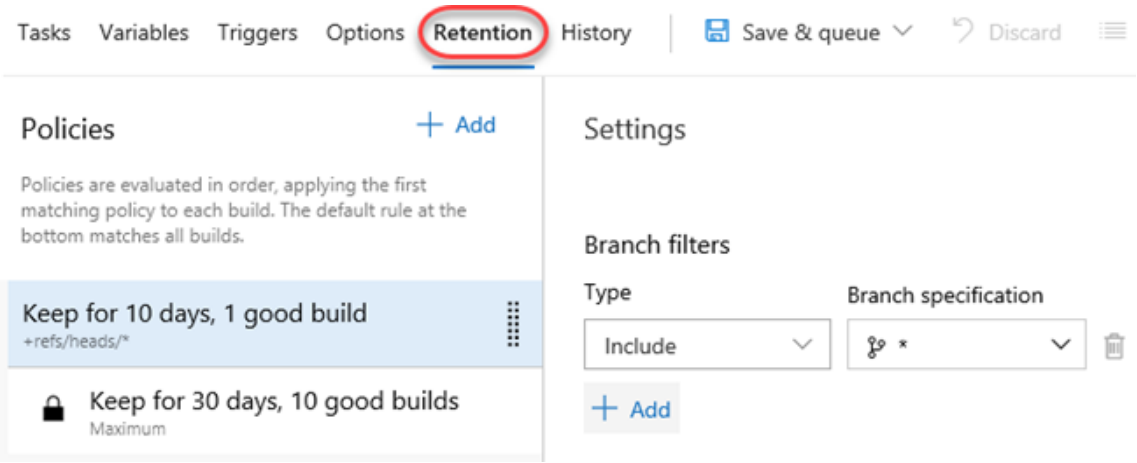
15. The **Triggers** tab enables you to define if and when builds are automatically invoked. Check **Enable continuous integration**, which will invoke a build when a change is committed to the master branch. You can also schedule builds, such as if your team runs nightly builds.



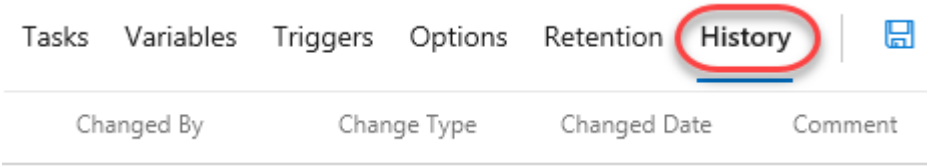
16. Select the **Options** tab. This provides a place to set build properties and limits.



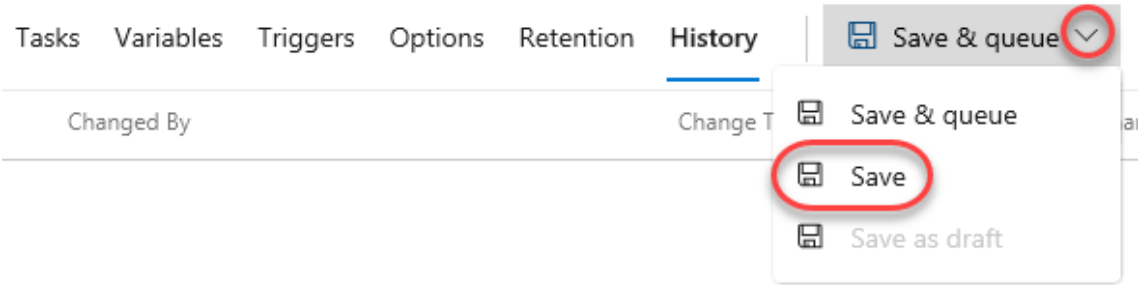
17. The **Retention** tab provides a place to set policies and settings for builds.



18. Select the **History** tab to view build history.

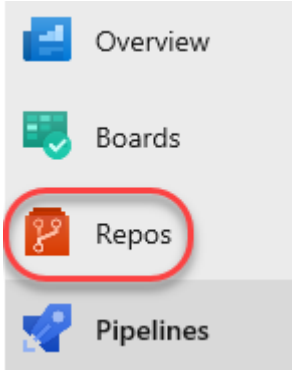


19. From the **Save & queue** dropdown, select **Save**.

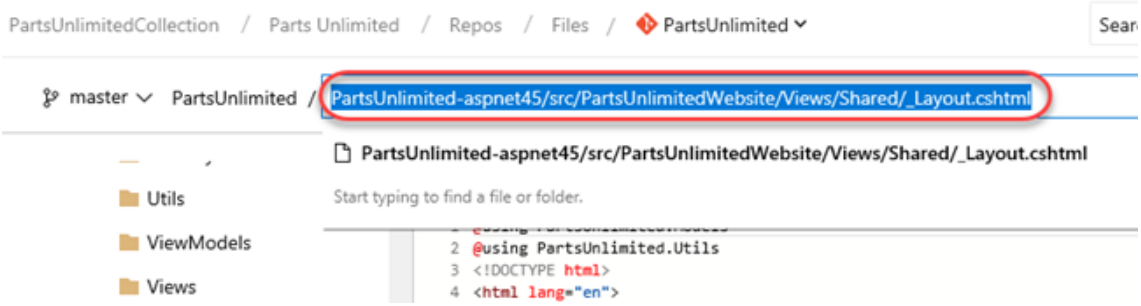


Task 2: Triggering and tracking a continuous integration build

1. Since the continuous integration trigger was enabled, you can invoke a build by committing a change to the master branch. Select **Repos**.

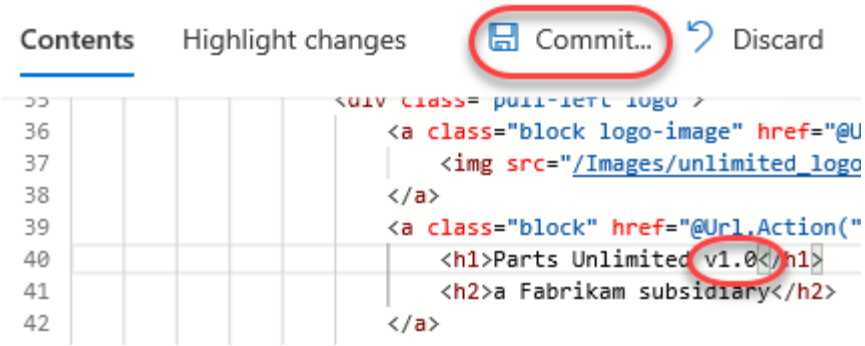


2. Navigate to **PartsUnlimited-aspnet45/src/PartsUnlimitedWebsite/Views/Shared/_Layout.cshtml**.

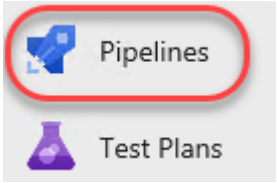


3. Click **Edit**.

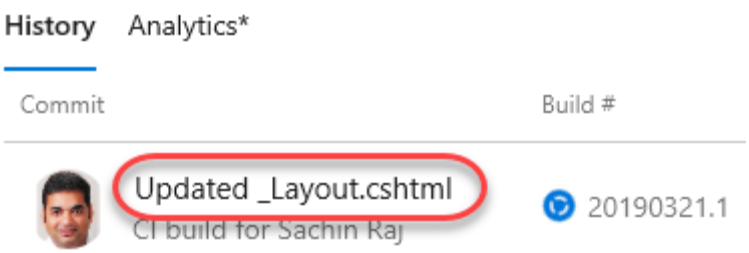
4. Make a cosmetic change by appending **"v1.0"** to the **h1** tag. Click **Commit** and confirm.



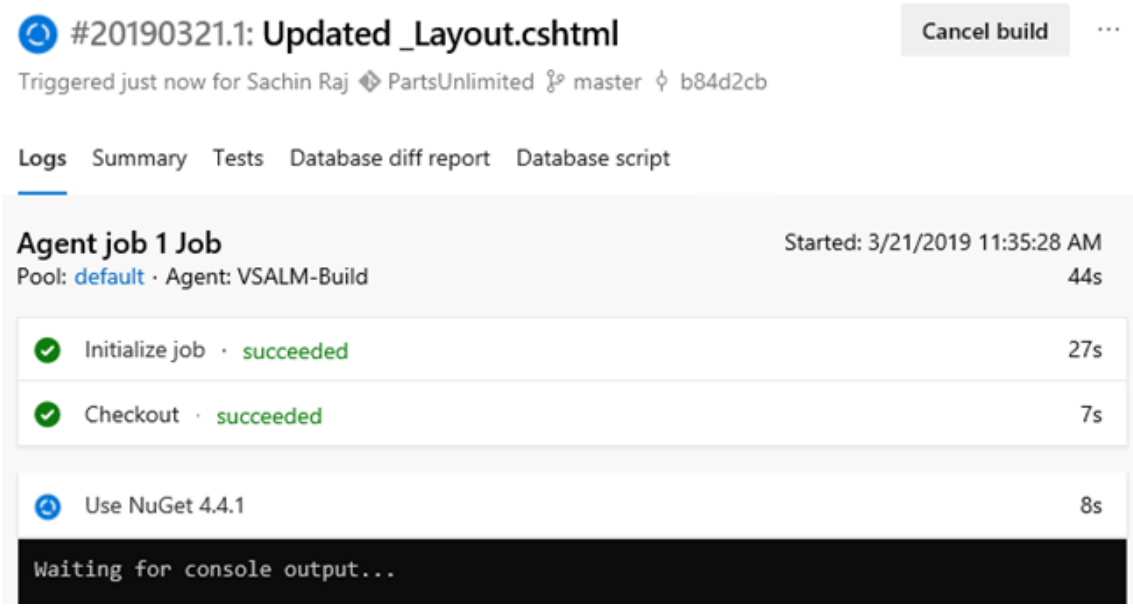
5. Select **Pipelines**.



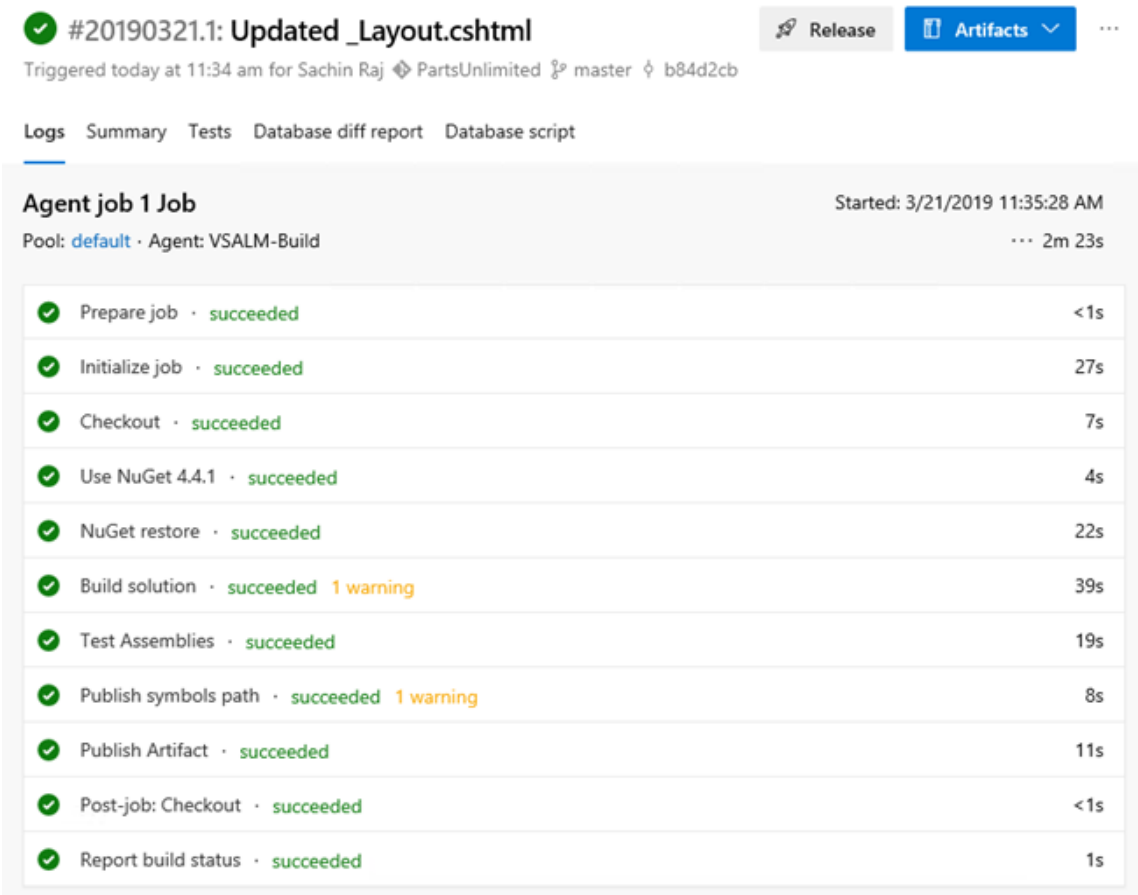
6. There should now be a build running. Select it.



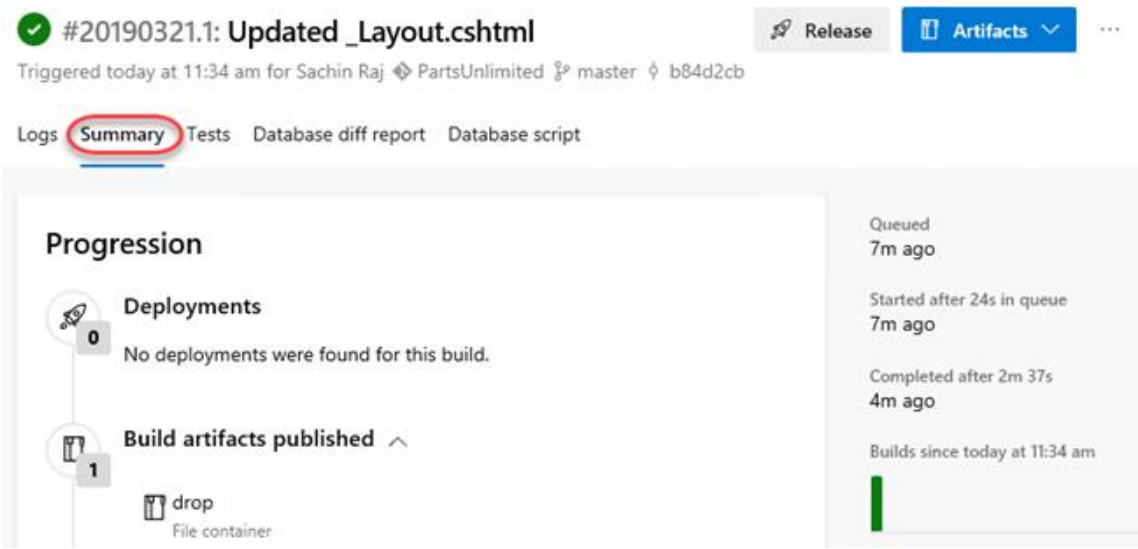
7. Follow the build through to completion. Each task is logged for easy tracking.



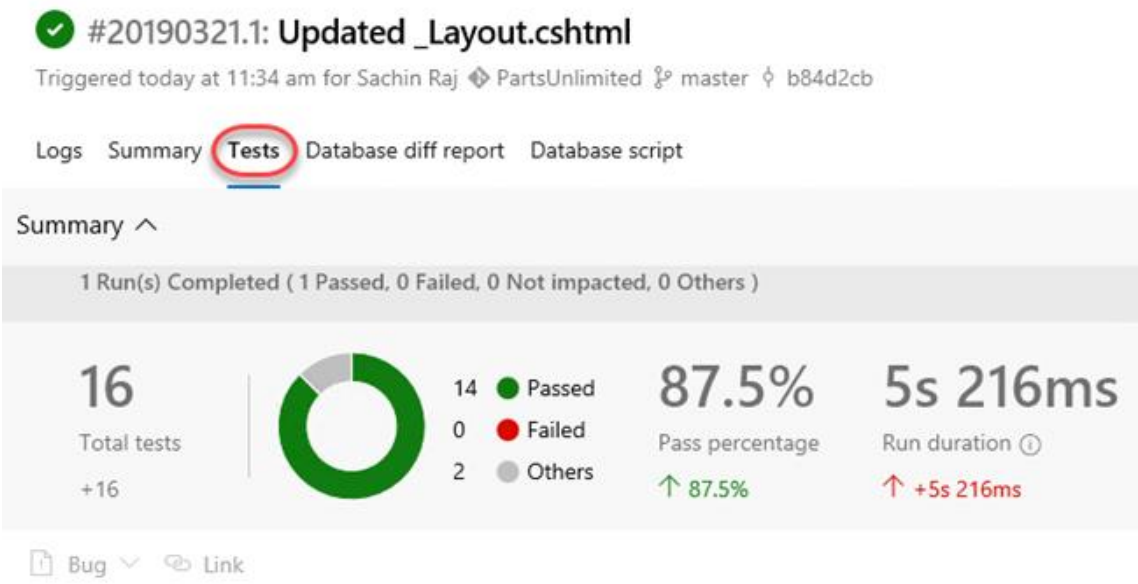
8. The build should succeed. If there are any errors, retrace earlier steps.



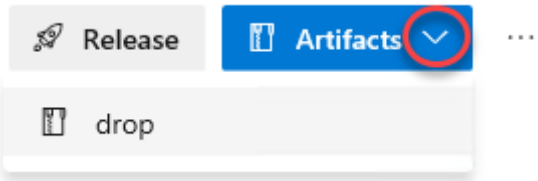
9. Select the **Summary** tab to review the progression of the lab.



10. The **Tests** tab provides a summary of the tests run during the build.



11. You can also review the build drop itself from the **Artifacts** dropdown.



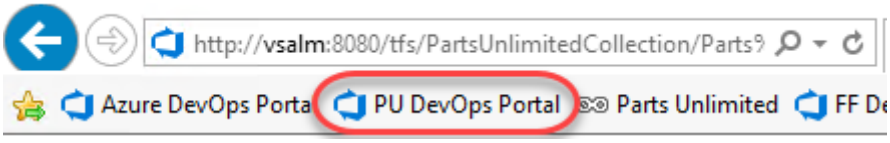
Exercise 2: Continuous Release Management

In this exercise, you will use the release management features of Azure DevOps Server to produce an automated deployment solution. This exercise will take an existing enterprise application and automate its deployment to the development team's testing environment after each source check-in.

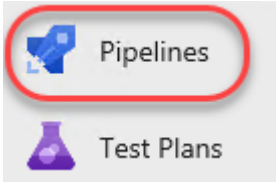
Task 1: Invoking a build

This task will create a build as a starting point for a continuous release. If you already completed the build lab, you can skip to the next task.

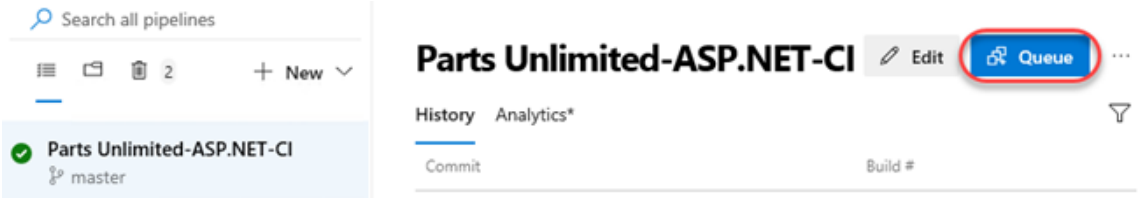
1. Log in as **Sachin Raj (VSALM\Sachin)**. All user passwords are **P2ssw0rd**.
2. Launch **Internet Explorer** from the taskbar and click **PU DevOps Portal** from the favorites bar at the top.



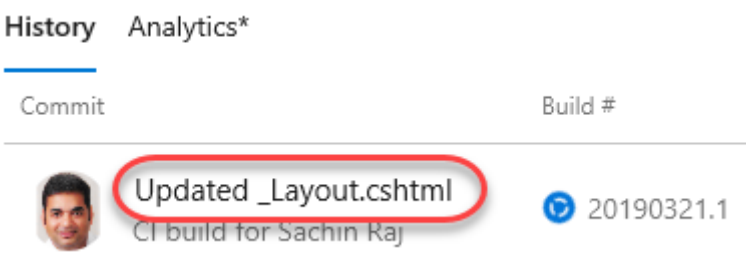
3. Select **Pipelines**.



4. Click **Queue** to invoke a build manually.



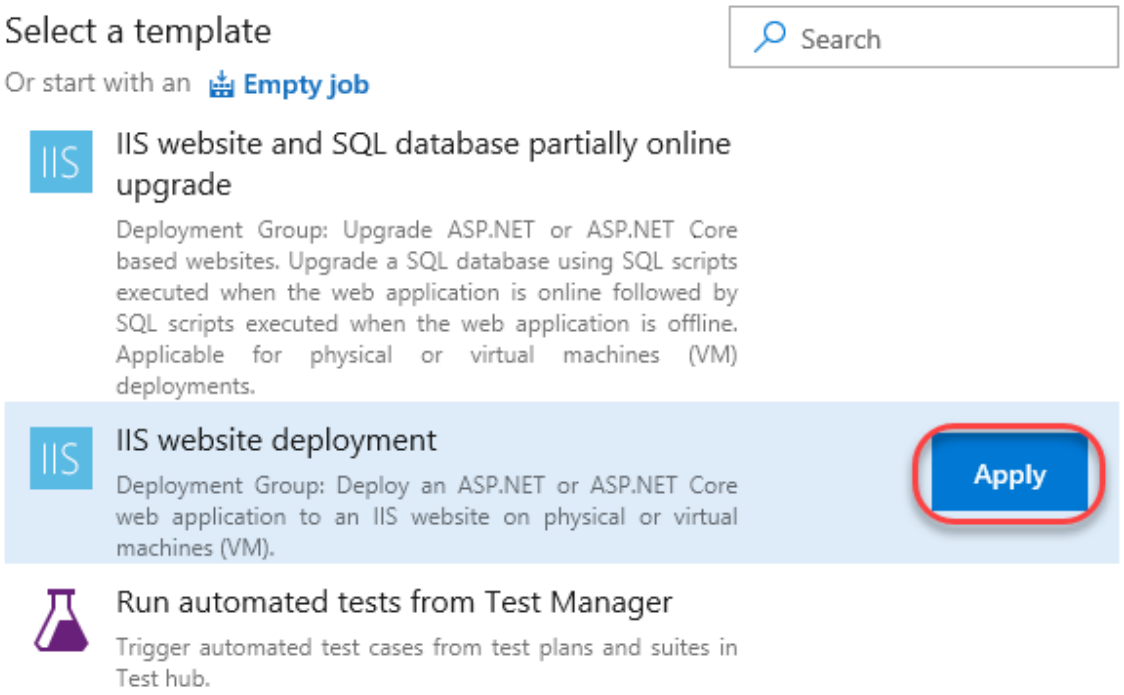
5. Select the build to follow its progress through to completion.



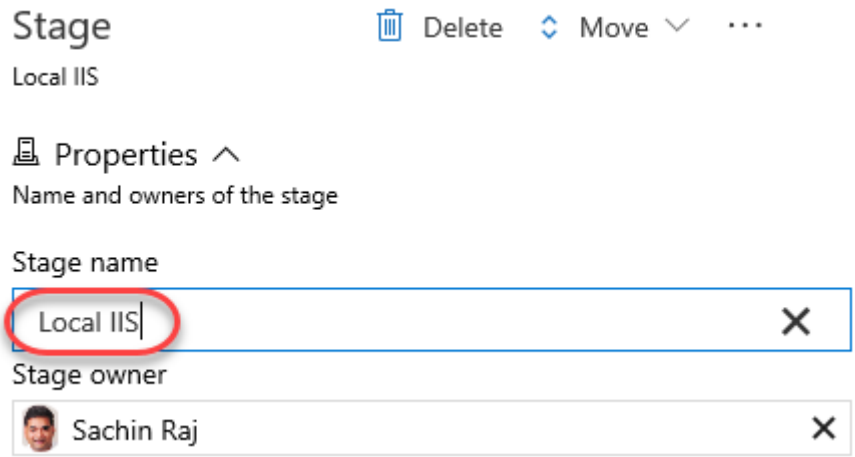
Task 2: Creating a continuous release pipeline

1. Once the build has completed, click **Release** to create a release pipeline. Note that you can also create a release pipeline from scratch, but this option will preconfigure the release pipeline to use this build pipeline's output.

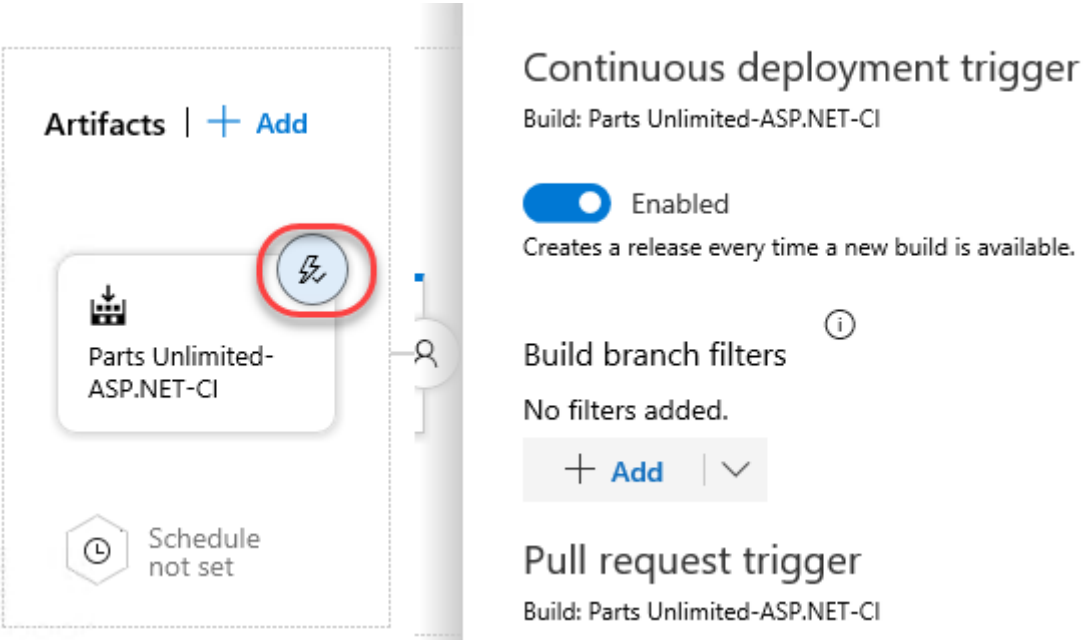
2. There are many release pipeline templates available out of the box for common deployment scenarios. To start off with, we will create a stage in the release pipeline that deploys the application to the IIS instance running on the VM. Select the **IIS website deployment** template and click **Apply**. This will provide the tasks required to deploy to IIS.



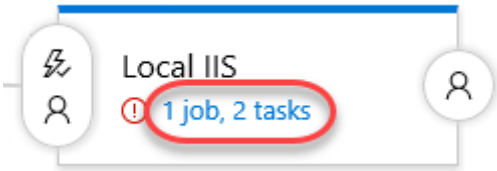
3. Set the **Stage name** to **"Local IIS"**.



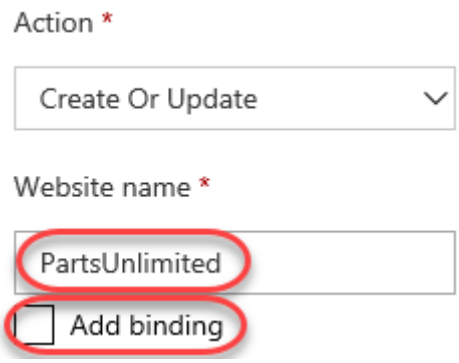
4. Since this release was created based on a successful build, the artifact details have been preconfigured. Click the **continuous deployment trigger** button on the **Artifacts** box to see that continuous deployment has been enabled so that every new build will invoke this release pipeline. And since the build pipeline is triggered by a master branch commit, any change from a developer can result in the site being updated with minimal overhead.



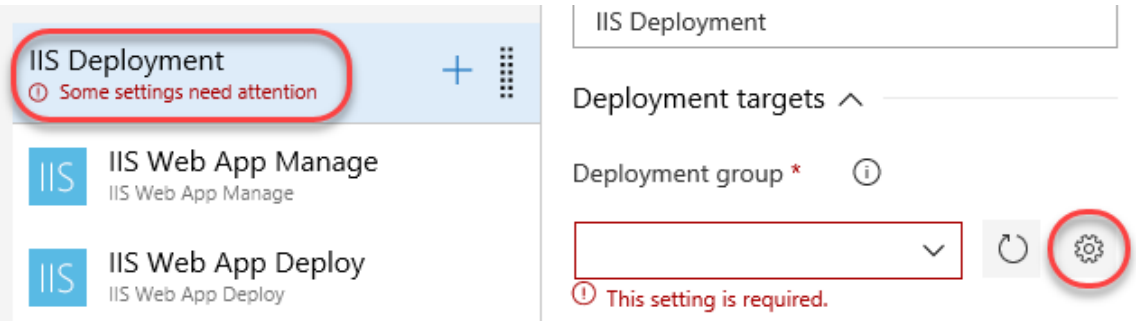
5. Select **1 job, 2 tasks** under the **Local IIS** stage.



6. The site we want to deploy to is hosted on the local IIS machine, so update the **Website name** to **"PartsUnlimited"**. Also clear the **Add binding** box since the bindings have already been configured.

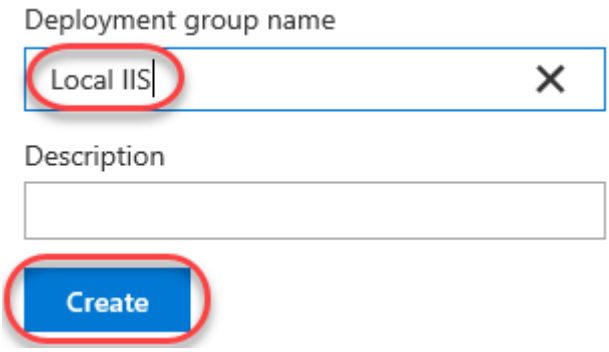


7. Select the **IIS Deployment** job definition. In order to deploy, we will need to define a **deployment group**. Click the **Settings** button to open this configuration in a new browser tab.



8. Click **Add a deployment group**.

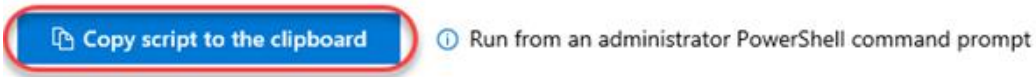
9. Set the **Deployment group name** to **"Local IIS"** and click **Create**.



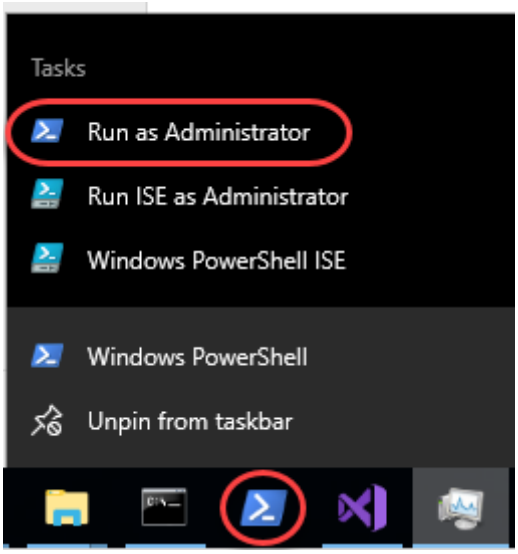
10. The provided PowerShell script will do everything you need to download, install, and configure the local machine as a deployment agent for this group. Click **Copy script to the clipboard**.

Registration script (PowerShell)

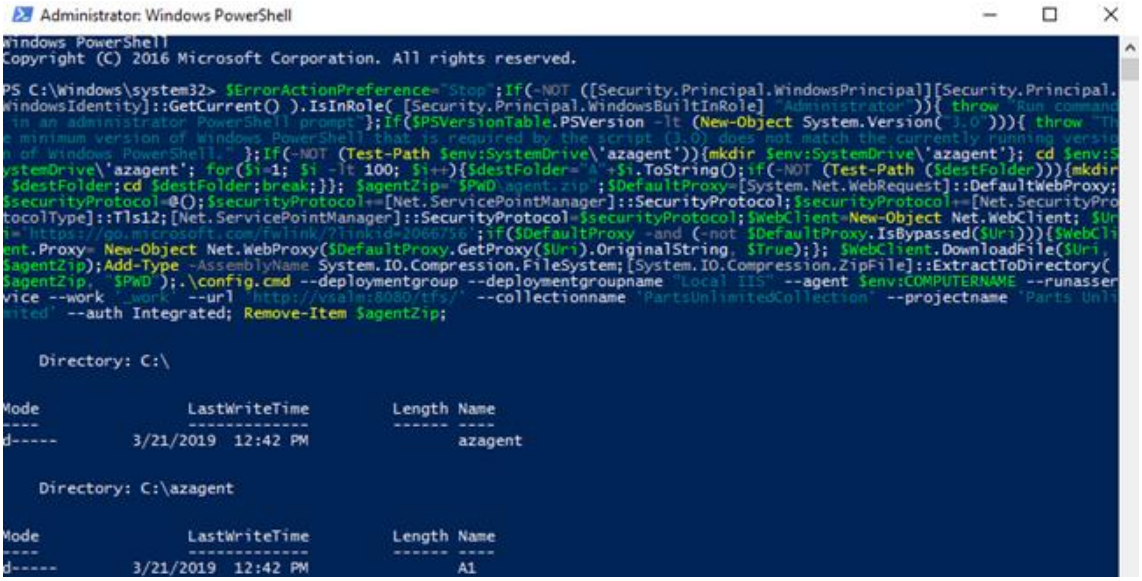
```
$ErrorActionPreference="Stop";If(-NOT ([Security.Principal.WindowsPrincipal]
[Security.Principal.WindowsIdentity]::GetCurrent() ).IsInRole
( [Security.Principal.WindowsBuiltInRole] "Administrator")){ throw "Run
command in an administrator PowerShell prompt"};If($PSVersionTable.PSVersion
-lt (New-Object System.Version("3.0"))){ throw "The minimum version of
Windows PowerShell that is required by the script (3.0) does not match the
currently running version of Windows PowerShell." };If(-NOT (Test-Path
$env:SystemDrive\'azagent')){mkdir $env:SystemDrive\'azagent'}; cd
$env:SystemDrive\'azagent'; for($i=1; $i -lt 100; $i++)
{$destFolder="A"+$i.ToString();if(-NOT (Test-Path ($destFolder))){mkdir
$destFolder;cd $destFolder;break;}}; $agentZip="$PWD\agent.zip";$DefaultProxy=
[System.Net.WebRequest]::DefaultWebProxy;$securityProtocol=@
();$securityProtocol+=
[Net.ServicePointManager]::SecurityProtocol;$securityProtocol+=
[Net.SecurityProtocolType]::Tls12;
[Net.ServicePointManager]::SecurityProtocol=$securityProtocol;$WebClient=New-
Object Net.WebClient; $Uri='https://go.microsoft.com/fwlink/?
linkid=2066756';if($DefaultProxy -and (-not $DefaultProxy.IsBypassed($Uri)))
{$WebClient.Proxy= New-Object Net.WebProxy($DefaultProxy.GetProxy
($Uri).OriginalString, $True);}; $WebClient.DownloadFile($Uri, $agentZip);Add-
Type -AssemblyName System.IO.Compression.FileSystem;
[System.IO.Compression.ZipFile]::ExtractToDirectory( $agentZip,
"$PWD");.\config.cmd --deploymentgroup --deploymentgroupname "Local IIS"
--agent $env:COMPUTERNAME --runasservice --work '_work' --url
'http://vsalm:8080/tfs/' --collectionname 'PartsUnlimitedCollection'
--projectname 'Parts Unlimited' --auth Integrated; Remove-Item $agentZip;
```



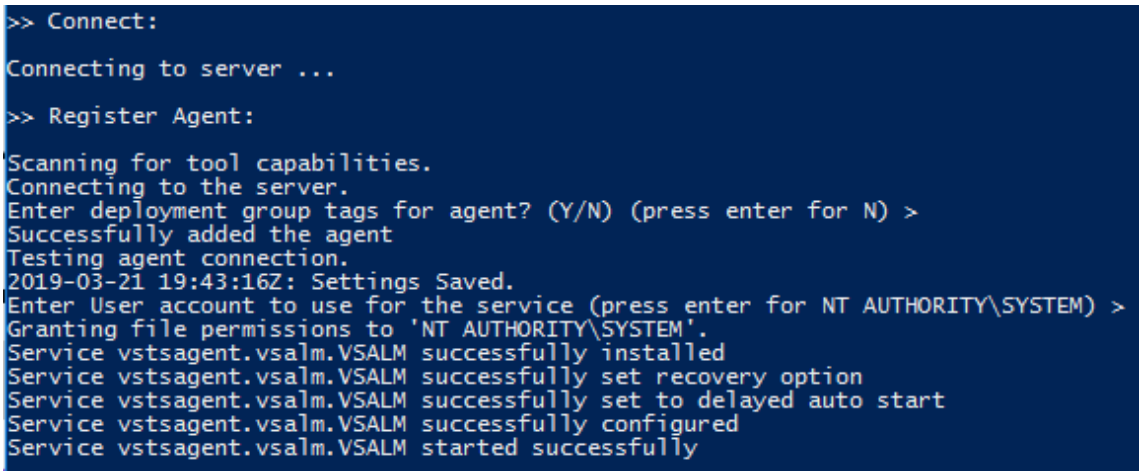
11. From the taskbar, right-click **PowerShell** and select **Run as Administrator**.



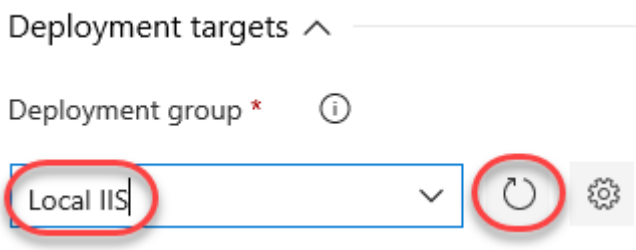
12. Execute the script from the clipboard. Don't forget to click **Enter** after pasting it.



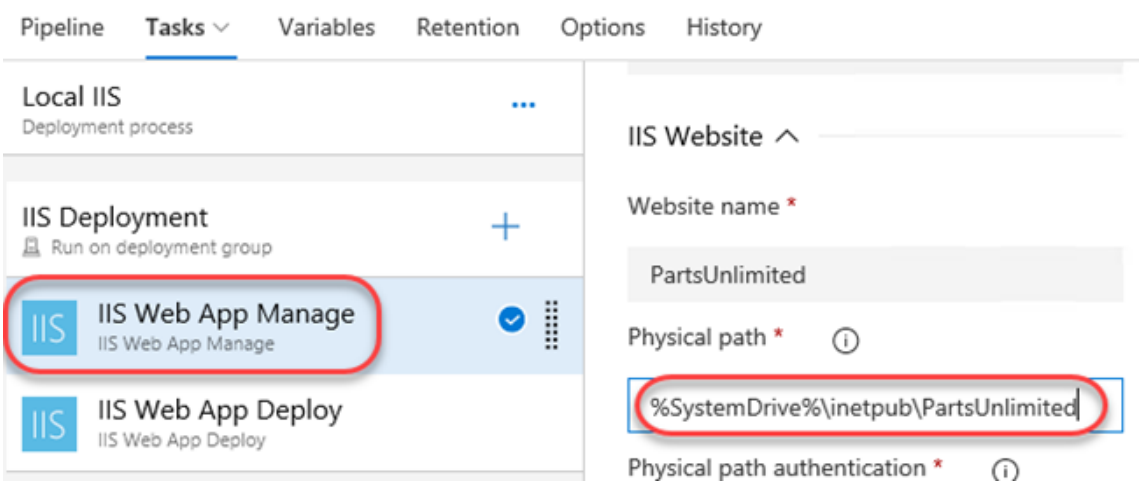
13. Accept the default options along the way. This will configure the agent to run as a service under the System account.



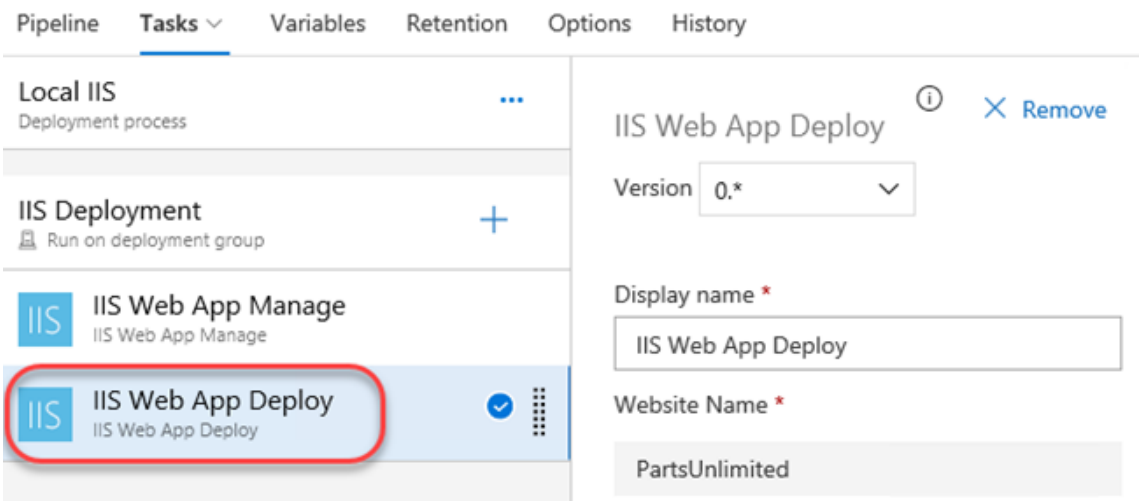
14. Close the deployment groups browser tab.
15. Return to the release pipeline and **Refresh** the deployment groups. Select **Local IIS**.



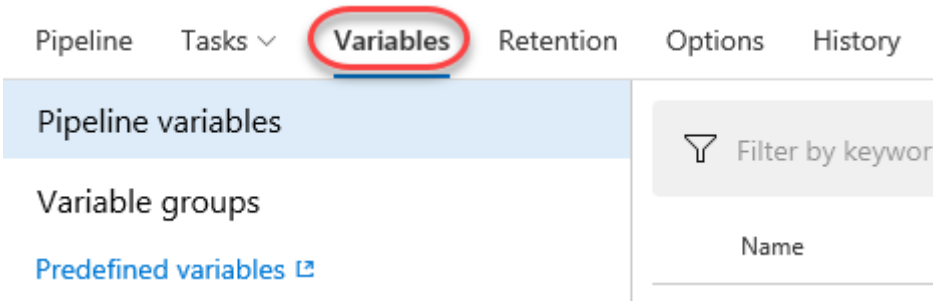
16. Select **IIS Web App Manage**. This is the first task in the job that defines properties such as the local path to deploy to. Update it to **"%SystemDrive%\inetpub\PartsUnlimited"**.



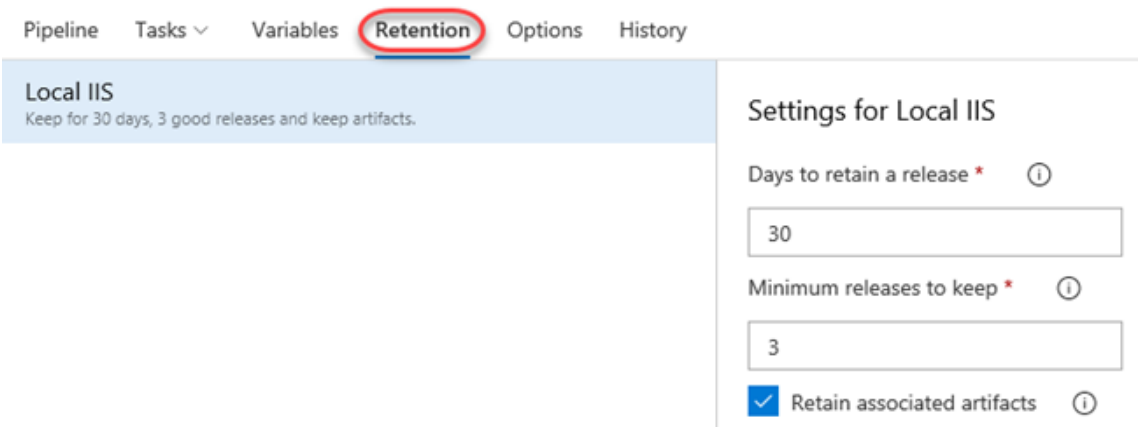
17. Select the **IIS Web App Deploy** tab. This task actually deploys the site.



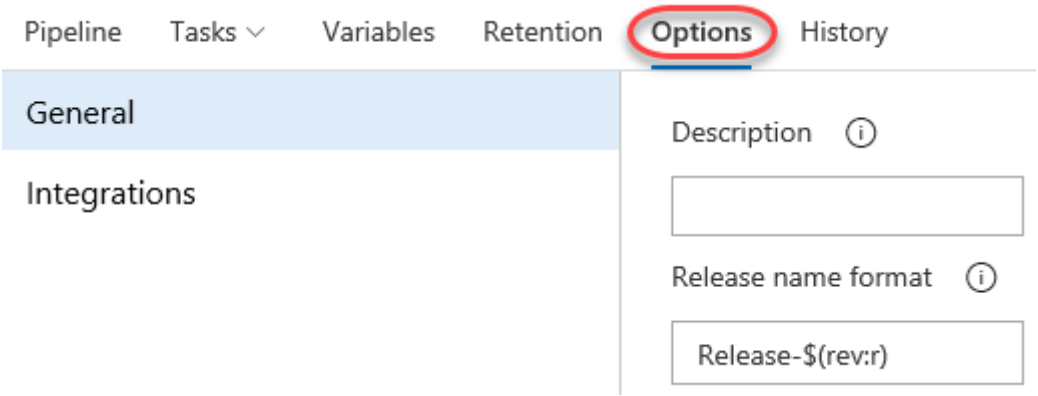
18. Select the **Variables** tab. This enables you to define pipeline-wide variables that can be managed centrally.



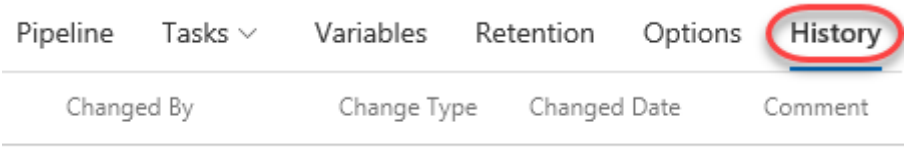
19. The **Retention** tab defines the policies for retaining releases.



20. The **Options** tab allows you to specify things like how releases are named.



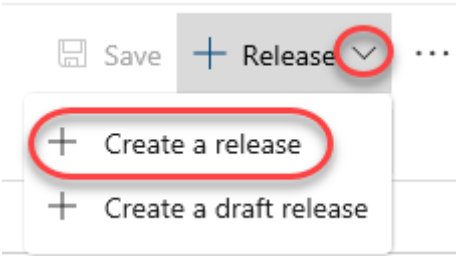
21. The **History** tab lists release history.



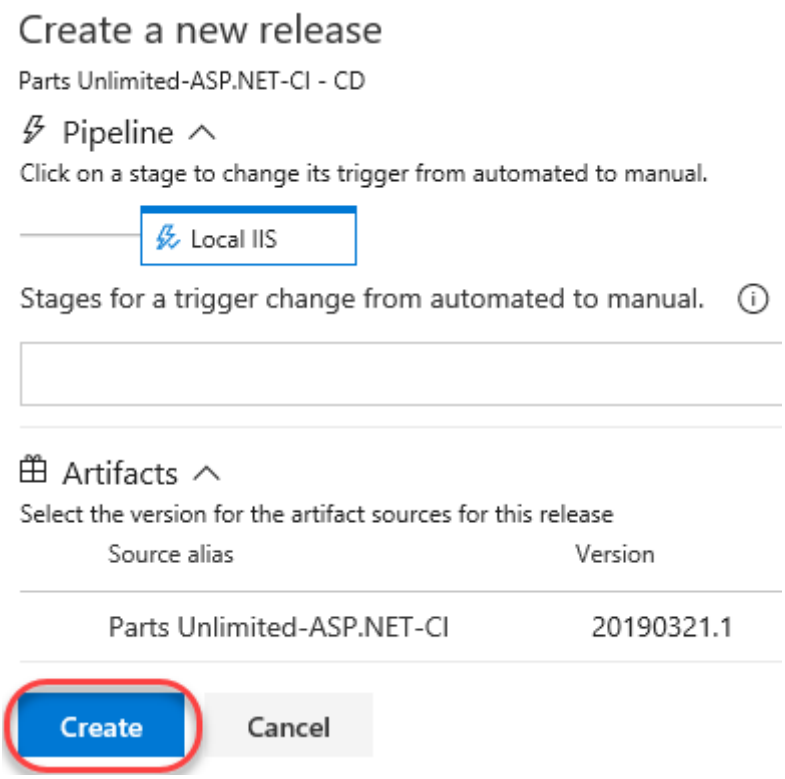
22. Click **Save** and confirm.

Task 3: Invoking a manual release

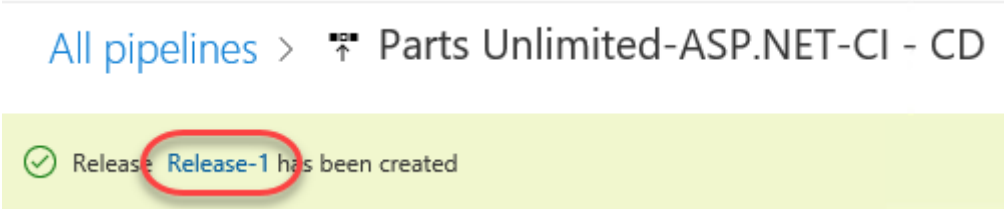
1. From the **Release** dropdown, select **Create a release**.



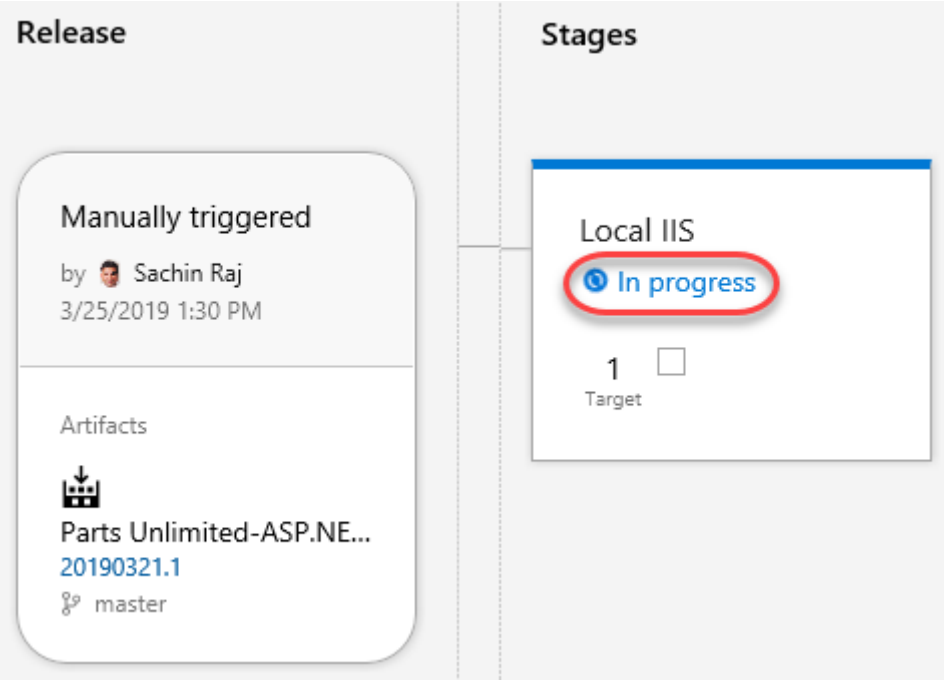
2. You have the option to override the behavior of the pipeline, which we won't do now. Click **Create** to begin the release using the latest build.



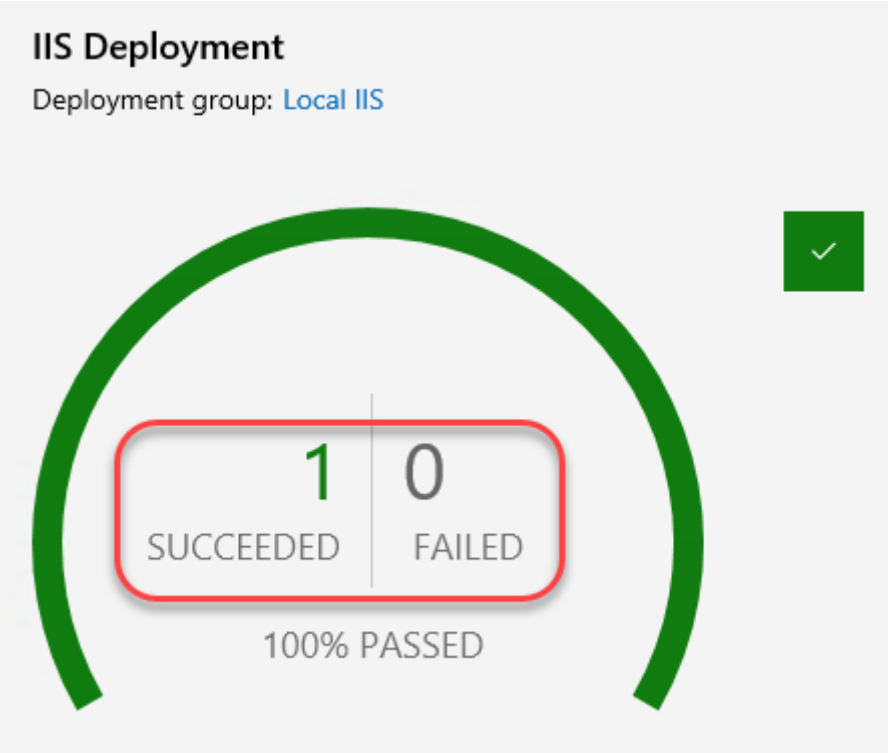
3. Click the new release to view it.



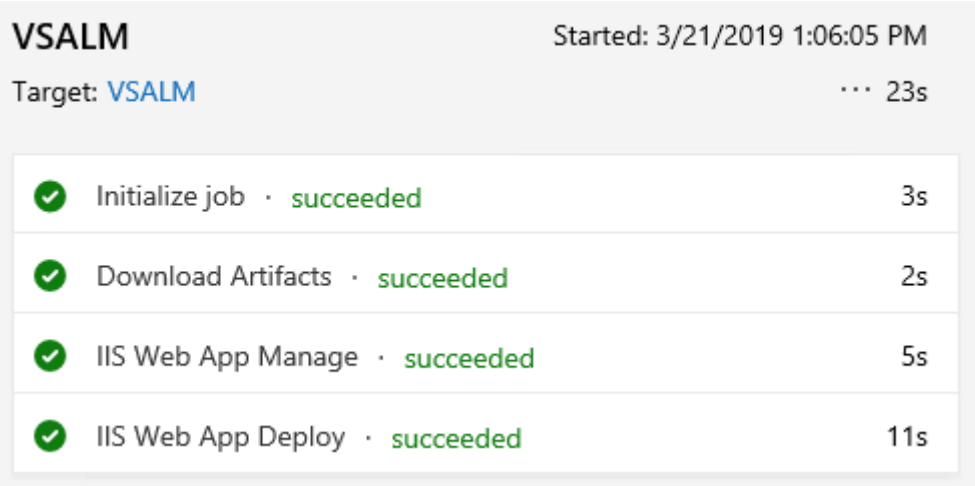
4. You can visualize the release through the pipeline using the same kind of view. Click **In progress**.



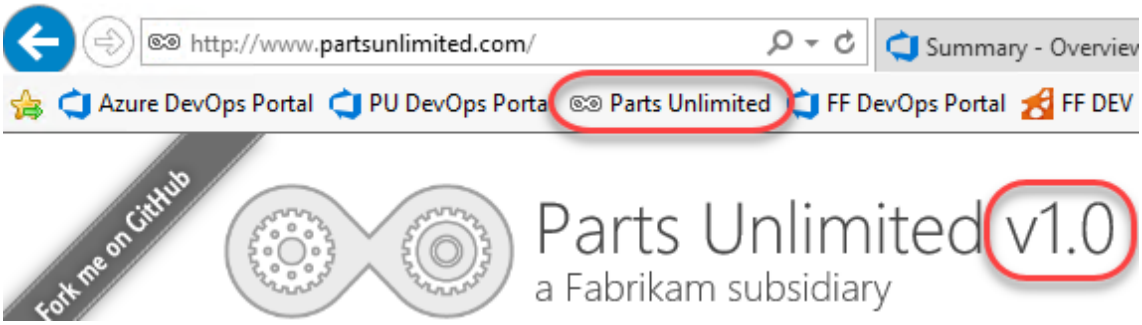
5. The release should complete pretty quickly since it's just a local deployment to IIS. Click the progress chart whether it's done or not.



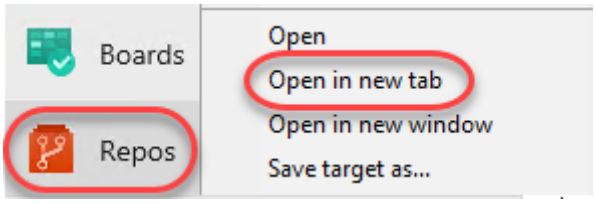
6. You can walk through each of the tasks to see the logs for what occurred.



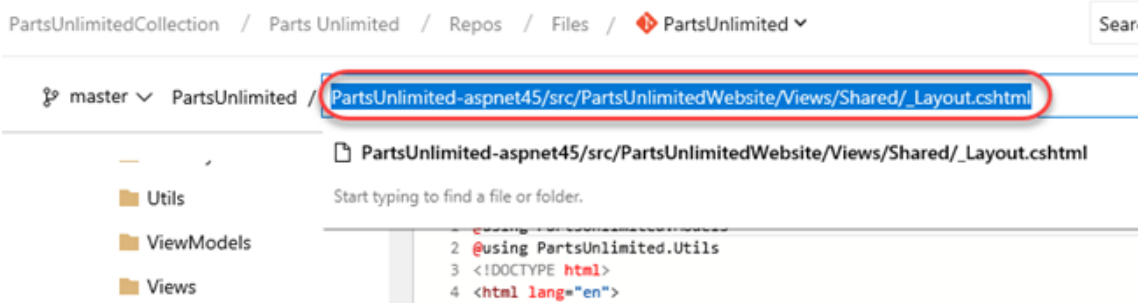
7. Open a new browser tab and click the **Parts Unlimited** site shortcut. If you made a visible change to the site, it should be apparent here.



8. Return to the release browser tab. Right-click **Repos** and select **Open in new tab**.

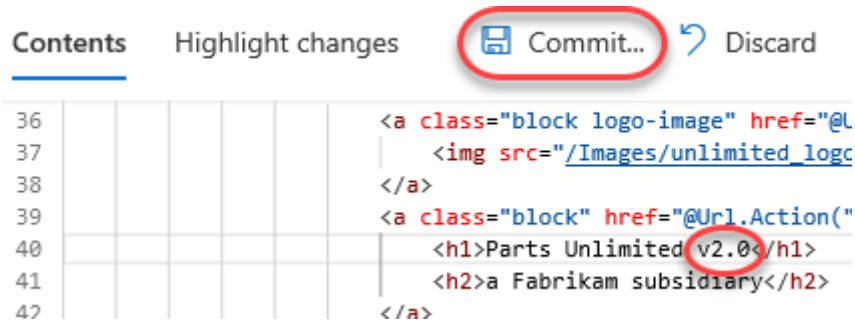


9. Navigate to **PartsUnlimited-aspnet45/src/PartsUnlimitedWebsite/Views/Shared/_Layout.cshtml**.

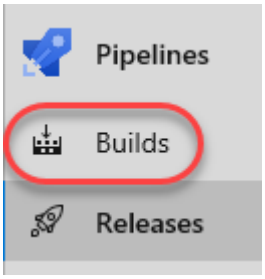


10. Click **Edit**.

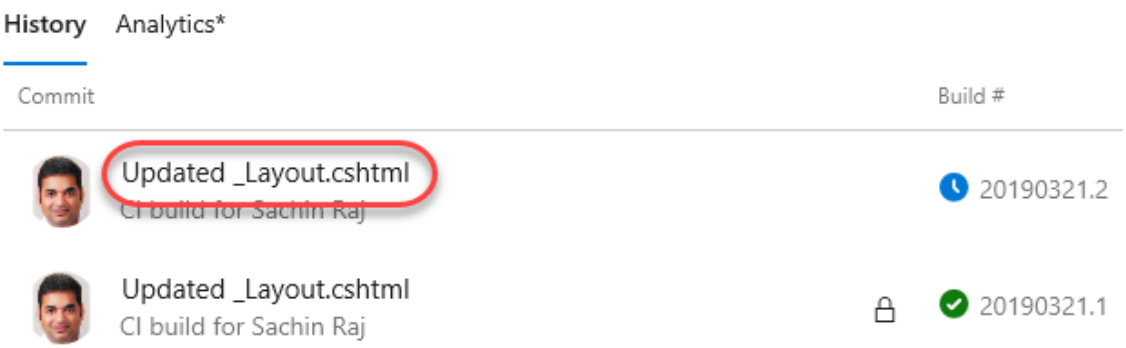
11. Make a cosmetic change by appending **"v2.0"** to the **h1** tag. Click **Commit** and confirm.



12. Return to the release browser tab and click **Builds**.

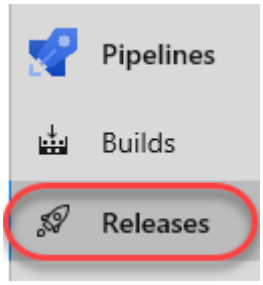


13. Click the newly launched build to follow it.

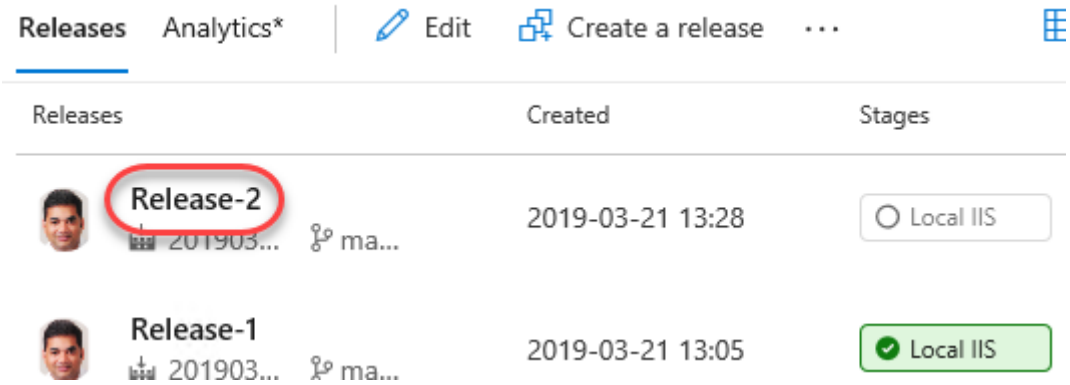


14. Wait for the build to complete.

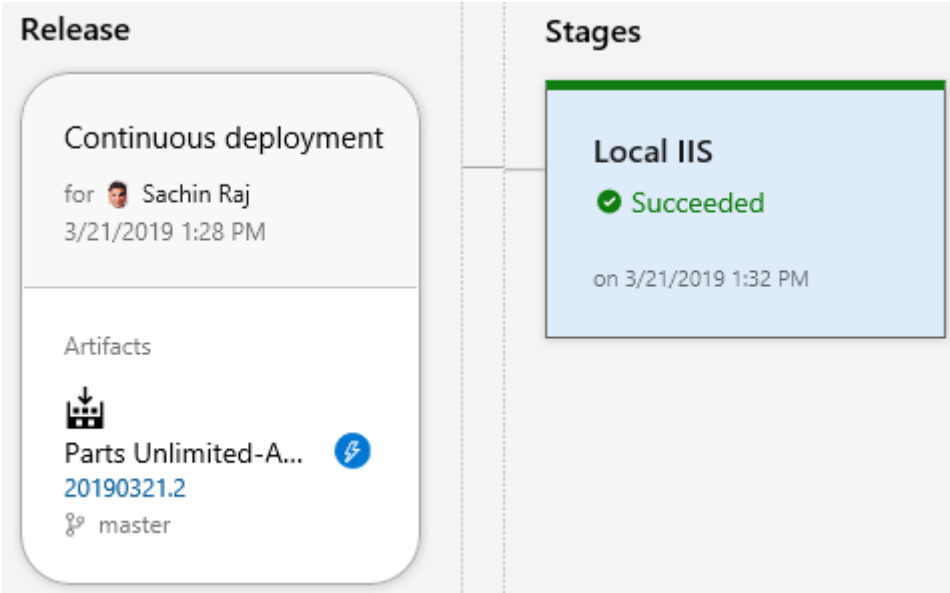
15. Navigate to the **Releases** view.



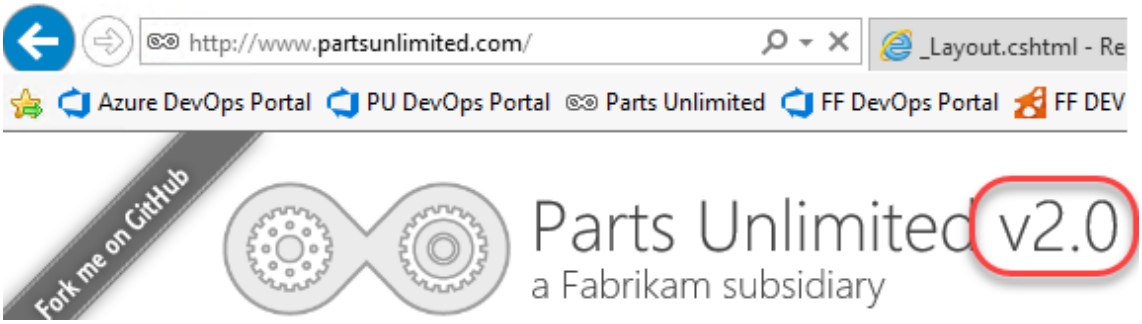
16. There should be a new release invoked by the completed build. Click it to open.



17. Like before, the release should quickly make its way through the pipeline and deploy to the local IIS.



18. Refresh the tab open to the Parts Unlimited site and note that the v2.0 is now visible.


http://www.partsunlimited.com/
_Layout.cshtml - Re
Azure DevOps Portal
PU DevOps Portal
Parts Unlimited
FF DevOps Portal
FF DEV
Fork me on GitHub
Parts Unlimited v2.0
a Fabrikam subsidiary